Software libre



Jesús González Barahona
Joaquín Seoane Pascual
Gregorio Robles

XP06/M2101/01492

Introducción al software libre

UOC
www.uoc.edu

David Megías Jiménez

Coordinador

Ingeniero en Informática por la UAB.

Magíster en Técnicas Avanzadas de Automatización de Procesos por la UAB.

Doctor en Informática por la UAB.

Profesor de los Estudios de Informática y Multimedia de la UOC.

Jordi Mas

Coordinador

Coordinador general de Softcatalà y desarrollador del procesador de textos libre Abiword.

Miembro fundador de Softcatalà y de la red telemática RedBBS.

En calidad de consultor, ha trabajado en empresas como Menta, Telépolis, Vodafone, Lotus, eresMas, Amena y Terra España.

Jesús González Barahona

Autor

Grupo de Sistemas y Comunicaciones, ESCET, Universidad Rey Juan Carlos de Madrid

jgb@gsyc.escet.urjc.es

Joaquín Seoane Pascual

Autor

Departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid

joaquin@dit.upm.es

Gregorio Robles

Autor

Grupo de Sistemas y Comunicaciones, ESCET, Universidad Rey Juan Carlos de Madrid

grex@gsyc.escet.urjc.es

Primera edición: noviembre 2003

Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-38.682-2003

Índice

Prólogo 11

Materiales previos 11
Agradecimientos 13

1. Introducción al software libre 15
1.1. El concepto de libertad en el software 15
1.1.1. Definición 16
1.1.2. Términos relacionados 17
1.2. Motivaciones 19
1.3. Consecuencias de la libertad del software 19
1.3.1. Para el usuario final 20
1.3.2. Para la Administración pública 21
1.3.3. Para el desarrollador 22
1.3.4. Para el integrador 22
1.3.5. Para el que proporciona mantenimiento y servicios 22
1.4. Otros recursos libres 23
1.4.1. Documentación de programas 23
1.4.2. Otra documentación 24
1.4.3. Artículos científicos 25
1.4.4. Leyes y estándares 25
1.4.5. Enciclopedias 28
1.4.6. Cursos 28
1.4.7. Colecciones y bases de datos 29
1.4.8. Hardware 30
1.4.9. Literatura y arte 30

2. Un poco de historia 31
2.1. El software libre antes del software libre 32
2.1.1. Y en el principio fue libre... 32
2.1.2. Años 1970 y primeros 1980 34
2.1.3. Desarrollo temprano de Unix 35
2.2. El comienzo: BSD, GNU 36
2.2.1. Richard Stallman, GNU, FSF: nace el movimiento del software libre 36

2.2.2. El CSRG de Berkeley 38
2.2.3. Los comienzos de Internet 40
2.2.4. Otros proyectos 43
2.3. Todo en marcha 43
2.3.1. En busca de un kernel 44
2.3.2. La familia *BSD 44
2.3.3. GNU/Linux entra en escena 45
2.4. Tiempos excitantes 47
2.4.1. Finales de 1990 47
2.4.2. Principios de los 2000 52
2.5. El futuro: ¿una carrera de obstáculos? 57
2.6. Algunas fechas de la historia del software libre 58

3. Aspectos legales 69
3.1. Breve introducción a la propiedad intelectual 69
3.1.1. Derechos de autor 70
3.1.2. Secreto comercial 72
3.1.3. Patentes y modelos de utilidad 73
3.1.4. Marcas y logotipos 74
3.2. Licencias en el software libre 75
3.2.1. Licencias tipo BSD 76
3.2.2. La Licencia Pública General de GNU (GNU GPL) 79
3.2.3. La Licencia Pública General Menor de GNU (GNU LGPL) 81
3.2.4. Otras licencias de programas 82
3.2.5. Distribución bajo varias licencias 86
3.3. Licencias de otros recursos libres 87
3.3.1. Licencia de documentación libre de GNU 87
3.3.2. Licencias de Creative Commons 88
3.4. Resumen 89

4. El desarrollador y sus motivaciones 91
4.1. Introducción 91
4.2. ¿Quiénes son los desarrolladores? 91
4.3. ¿Qué hacen los desarrolladores? 93
4.4. Distribución geográfica 94
4.5. Dedicación 96
4.6. Motivaciones 98
4.7. Liderazgo 99
4.8. Resumen y conclusiones 102

5. Economía 103

5.1. Financiación de proyectos de software libre 103
5.1.1. Financiación pública 104
5.1.2. Financiación privada sin ánimo de lucro 107
5.1.3. Financiación por quien necesita mejoras 108
5.1.4. Financiación con beneficios relacionados 109
5.1.5. Financiación como inversión interna 111
5.1.6. Otros modos de financiación 113
5.2. Modelos de negocio basados en software libre 116
5.2.1. Mejor conocimiento 118
5.2.2. Mejor conocimiento con limitaciones 120
5.2.3. Fuente de un producto libre 122
5.2.4. Fuente de un producto con limitaciones 124
5.2.5. Licencias especiales 126
5.2.6. Venta de marca 127
5.3. Otras clasificaciones de modelos de negocio 127
5.3.1. Clasificación de Hecker 128
5.4. Impacto sobre las situaciones de monopolio 129
5.4.1. Elementos que favorecen los productos dominantes 130
5.4.2. El mundo del software propietario 131
5.4.3. La situación con software libre 132
5.4.4. Estrategias para constituirse en monopolio con software libre 134

6. Iniciativas públicas 137
6.1. Impacto del software libre en las administraciones públicas 137
6.1.1. Impactos principales 138
6.1.2. Dificultades de adopción 142
6.2. Actuaciones de las administraciones públicas en el mundo del software 144
6.2.1. ¿Cómo satisfacer mejor las necesidades de las administraciones públicas? 145
6.2.2. Promoción de software 148
6.2.3. Fomento de la investigación 150
6.3. Iniciativas legislativas 151
6.3.1. Proyectos de ley en Francia 151
6.3.2. Proyecto de ley en Brasil 152
6.3.3. Proyectos de ley en Perú 153
6.3.4. Proyectos de ley en España 155
6.4. Textos de algunas propuestas legislativas y documentos relacionados 156
6.4.1. Proyecto de ley de Laffitte, Trégouet y Cabanel (Francia) 156

6.4.2. Proyecto de ley de Le Déaut, Paul y Cohen (Francia) ... 159
6.4.3. Proyecto de ley de Villanueva y Rodrich (Perú) ... 164
6.4.4. Cartas de Microsoft Perú y del congresista Villanueva ... 177
6.4.5. Decreto de medidas de impulso de la sociedad del conocimiento en Andalucía ... 197

7. Ingeniería del software libre ... 199
7.1. Introducción ... 199
7.2. "La catedral y el bazar" ... 200
7.3. Liderazgo y toma de decisiones en el bazar ... 203
7.4. Procesos en el software libre ... 205
7.5. Crítica a "La catedral y el bazar" ... 207
7.6. Estudios cuantitativos ... 208
7.7. Trabajo futuro ... 212
7.8. Resumen ... 214

8. Entornos y tecnologías de desarrollo ... 217
8.1. Caracterización de entornos, herramientas y sistemas ... 217
8.2. Lenguajes y herramientas asociadas ... 218
8.3. Mecanismos básicos de colaboración ... 219
8.4. Gestión de fuentes ... 221
8.4.1. CVS ... 222
8.4.2. Otros sistemas de gestión de fuentes ... 226
8.5. Documentación ... 228
8.5.1. Docbook ... 230
8.5.2. Wikis ... 231
8.6. Gestión de errores ... 233
8.7. Sistemas de gestión de flujo de trabajos ... 234
8.8. Soporte para otras arquitecturas ... 235
8.9. Sitios de soporte al desarrollo ... 236
8.9.1. SourceForge ... 236
8.9.2. Herederos de SourceForge ... 239
8.9.3. Otros sitios y programas ... 239

9. Estudio de casos ... 241
9.1. Linux ... 243
9.1.1. Historia de Linux ... 243
9.1.2. El modo de trabajo de Linux ... 245
9.1.3. Estado actual de Linux ... 247

9.2. FreeBSD ... 249
9.2.1. Historia de FreeBSD ... 250
9.2.2. Desarrollo en FreeBSD ... 250
9.2.3. Toma de decisiones en FreeBSD ... 251
9.2.4. Empresas alrededor de FreeBSD ... 252
9.2.5. Estado actual de FreeBSD ... 253
9.2.6. Radiografía de FreeBSD ... 253
9.2.7. Estudios académicos sobre FreeBSD ... 256
9.3. KDE ... 256
9.3.1. Historia de KDE ... 257
9.3.2. Desarrollo de KDE ... 258
9.3.3. La Liga KDE ... 259
9.3.4. Estado actual de KDE ... 261
9.3.5. Radiografía de KDE ... 262
9.4. GNOME ... 265
9.4.1. Historia de GNOME ... 265
9.4.2. La Fundación GNOME ... 266
9.4.3. La industria alrededor de GNOME ... 268
9.4.4. Estado actual de GNOME ... 270
9.4.5. Radiografía de GNOME ... 271
9.4.6. Estudios académicos sobre GNOME ... 274
9.5. Apache ... 274
9.5.1. Historia de Apache ... 274
9.5.2. Desarrollo de Apache ... 276
9.5.3. Radiografía de Apache ... 277
9.6. Mozilla ... 278
9.6.1. Historia de Mozilla ... 279
9.6.2. Estado actual de Mozilla ... 282
9.6.3. Radiografía de Mozilla ... 283
9.7. OpenOffice.org ... 284
9.7.1. Historia de OpenOffice.org ... 285
9.7.2. Organización de OpenOffice.org ... 286
9.7.3. Radiografía de OpenOffice.org ... 286
9.8. Red Hat Linux ... 288
9.8.1. Historia de Red Hat ... 288
9.8.2. Estado actual de Red Hat ... 290
9.8.3. Radiografía de Red Hat ... 291
9.9. Debian GNU/Linux ... 293
9.9.1. Radiografía de Debian ... 295
9.9.2. Comparación con otros sistemas operativos . 298
Bibliografía ... 301

Appendix A. GNU Free Documentation License 317

A.1. PREAMBLE 317
A.2. APPLICABILITY AND DEFINITIONS 318
A.3. VERBATIM COPYING 320
A.4. COPYING IN QUANTITY 320
A.5. MODIFICATIONS 321
A.6. COMBINING DOCUMENTS 324
A.7. COLLECTIONS OF DOCUMENTS 324
A.8. AGGREGATION WITH INDEPENDENT WORKS 325
A.9. TRANSLATION 325
A.10. TERMINATION 326
A.11. FUTURE REVISIONS OF THIS LICENSE 326
A.12. ADDENDUM: How to use this License for your documents 326

Apéndice B. Licencia de Documentación Libre de GNU 329

B.1. PREÁMBULO 329
B.2. APLICABILIDAD Y DEFINICIONES 330
B.3. COPIA LITERAL 332
B.4. COPIADO EN CANTIDAD 334
B.5. MODIFICACIONES 336
B.6. COMBINACIÓN DE DOCUMENTOS 337
B.7. COLECCIONES DE DOCUMENTOS 338
B.8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES 338
B.9. TRADUCCIÓN 338
B.10. TERMINACIÓN 338
B.11. REVISIONES FUTURAS DE ESTA LICENCIA 339
B.12. ADENDA: Cómo usar esta Licencia en sus documentos 339

Lista de tablas

4-1. Países con mayor número de desarrolladores de Debian ... 95
4-2. Dedicación en horas semanales ... 97
4-3. Grado de conocimiento de desarrolladores importantes ... 100
9-1. Estado actual de Linux ... 248
9-2. Lenguajes de programación utilizados en Linux ... 249
9-3. Estado actual de FreeBSD ... 255
9-4. Lenguajes de programación utilizados en FreeBSD ... 256
9-5. Estado actual de KDE ... 263
9-6. Lenguajes de programación utilizados en KDE ... 264
9-7. Estado actual de GNOME ... 272
9-9. Estado actual de Apache ... 278
9-10. Lenguajes de programación utilizados en Apache ... 278
9-11. Estado actual de Mozilla ... 283
9-12. Lenguajes de programación utilizados en Mozilla ... 284
9-13. Estado actual de OpenOffice.org ... 287
9-14. Lenguajes de programación utilizados en OpenOffice.org ... 287
9-15. Estado actual de Red Hat Linux ... 292
9-16. Lenguajes de programación utilizados en Red Hat ... 293
9-17. Estado actual de Debian ... 296
9-18. Lenguajes de programación utilizados en Debian GNU/Linux ... 297
9-19. Lenguajes más utilizados en Debian ... 297
9-20. Comparación con sistemas propietarios ... 299

Prólogo

La primera versión de estos apuntes fue escrita por Jesús M. González Barahona, Joaquín Seoane Pascual y Gregorio Robles entre los meses de abril y septiembre del 2003. Aunque llevábamos tiempo hablando sobre preparar un material de este tipo para la asignatura Software Libre que Joaquín y Jesús impartimos en los programas de doctorado de nuestros respectivos departamentos, la iniciativa fue de la Universitat Oberta de Catalunya (UOC) de encargarnos un material para la asignatura de introducción a su máster de software libre lo que nos decidió finalmente a ponernos manos a la obra. En este encargo fue fundamental la labor de Jordi Mas, coordinador académico del máster, que no sólo nos propuso para este trabajo y nos puso en contacto con la UOC, sino que nos acompañó en las relaciones con ellos durante el proyecto.

Materiales previos

Algunos textos de estos apuntes están basados en materiales previos, normalmente de los propios autores, en algunos casos de terceras personas (utilizados con permiso, cuando no han sido completamente reelaborados). Entre ellos podemos mencionar los siguientes (a riesgo de olvidar alguno importante):

- Hay algunos fragmentos (sobre todo en los capítulos de historia y economía) inspirados en el documento "Free Software / Open Source: Information Society Opportunities for Europe?", que Jesús González Barahona coeditó para la Comisión Europea. Sin embargo, los fragmentos en cuestión han sido ampliados, retocados y actualizados, tanto que en muchos casos pueden ser difíciles de reconocer.

Lectura complementaria

[Ver bibliografía, apartado 111]

- El apartado sobre los monopolios y el software libre está reelaborado basándonos en el artículo "Software libre, monopolios y otras yerbas", de Jesús M. González Barahona.

Lectura complementaria

[Ver bibliografía, apartado 66]

Lectura complementaria

[Ver bibliografía, apartado 84]

Lectura complementaria

[Ver bibliografía, apartado 67]

- Los apartados sobre iniciativas legislativas e iniciativas de administraciones públicas en relación con el software libre están en parte basados en "Iniciativas de las administraciones públicas en relación al Software Libre" (gracias a Pedro de las Heras por permitirnos utilizar ese material, del que es coautor).

- Parte del apartado sobre los motivos para usar software libre en las administraciones públicas está basado en el artículo de Jesús M. González Barahona.

- La traducción de la Licencia de Documentación Libre de GNU es una actualización adaptada de la realizada por Igor Támara y Pablo Reyes para la versión 1.1, a los que agradecemos el haberla realizado y su permiso para modificarla.

- El capítulo de ingeniería del software libre es una adaptación de un artículo sobre el estado del arte de la ingeniería del software aplicada al software libre de Jesús M. González Barahona y Gregorio Robles para la revista Novática.

- En el capítulo de estudios de casos, la parte dedicada al desarrollo de Linux se basa en una presentación que realizó Juan-Mariano de Goyeneche durante el curso de doctorado "Programas Libres" de la Universidad Politécnica de Madrid durante el curso 2002-03.

- La parte histórica del estudio pormenorizado de GNOME ha sido tomada de la introducción histórica incluida en el libro *Desarrollo de aplicaciones en GNOME2* elaborado por GNOME Hispano y realizada por uno de los autores de este libro.

- El caso de estudio de FreeBSD se basa en parte en la ponencia presentada por Jesús Rodríguez en el III Congreso HispaLinux celebrado en Madrid en el año 2000.

- Los casos de estudio de Debian y Red Hat parten del trabajo previo de González Barahona et al. que han plasmado en varios artículos los resultados del análisis cuantitativo de estas dos distribuciones.

ANOTACIONES

Agradecimientos

Los autores agradecen a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la primera edición de esta obra, enmarcada en el Máster Internacional en Software Libre ofrecido por la citada institución.

ANOTACIONES

1. Introducción al software libre

¿Qué es el software libre? ¿Qué es y qué implicaciones tiene la licencia de un programa libre? ¿Cómo se está desarrollando el software libre? ¿Cómo se financian los proyectos de software libre, qué modelos de negocio se están experimentando relacionados con ellos? ¿Qué motiva a los desarrolladores, especialmente a los que son voluntarios, a involucrarse en proyectos de software libre? ¿Cómo son estos desarrolladores? ¿Cómo se coordinan en sus proyectos, y cómo es el software que producen? En resumen, ¿cuál es la panorámica general del software libre? Éste es el tipo de preguntas que trataremos de responder en este texto. Porque aunque el software libre está cada vez más en los medios de comunicación, en las conversaciones de los profesionales de la informática, e incluso empieza a estar en boca de los ciudadanos en general, aún es un desconocido para muchos. Y muchas veces los que lo conocen lo hacen sólo de algunos de sus aspectos, desconociendo completamente otros.

Para empezar, en este capítulo vamos a presentar los aspectos específicos del software libre, centrándonos fundamentalmente en explicar sus bases para los que se aproximen al tema por primera vez, y en motivar su importancia. Entre estas bases nos detendremos en la definición del término (para saber de qué vamos a hablar) y en las consecuencias principales del uso (y la mera existencia) del software libre. También presentaremos otros recursos libres distintos del software y que han ido apareciendo, en parte, bajo el impulso y ejemplo de aquél.

1.1. El concepto de libertad en el software

Desde hace más de 30 años nos hemos acostumbrado a que quien me vende un programa me impone las condiciones bajo las que puedo usarlo, prohibiéndome, por ejemplo, que se lo pase a un amigo. A pesar de ser software, no puedo adaptarlo a mis necesidades, ni siquiera corregir errores, debiendo esperar a que el fabricante los arre-

ANOTACIONES

gle. Esto no tiene por qué ser así, y es precisamente el software libre el que me concede las libertades que el software propietario me niega.

1.1.1. Definición

Lectura complementaria

[Ver bibliografía, apartado 98]

Así pues el término software libre (o programas libres) se refiere a libertad, tal como fue concebido por Richard Stallman en su definición. En concreto se refiere a cuatro libertades:

1. Libertad para ejecutar el programa en cualquier sitio, con cualquier propósito y para siempre.
2. Libertad para estudiarlo y adaptarlo a nuestras necesidades. Esto exige el acceso al código fuente.
3. Libertad de redistribución, de modo que se nos permita colaborar con vecinos y amigos.
4. Libertad para mejorar el programa y publicar las mejoras. También exige el código fuente.

Estas libertades se pueden garantizar de acuerdo con la legalidad vigente por medio de una licencia, como veremos en el capítulo 3. En ella se plasman las libertades, pero también restricciones compatibles con ellas, como dar crédito a los autores originales si redistribuimos. Incluso puede obligarnos a que los programas ajenos mejorados por nosotros también sean libres, promoviendo así la creación de más software libre.



La ambigüedad de "free"

El término original en inglés para 'programas libres' es *free software*. Sin embargo, en inglés el término *free* además de 'libre' significa 'gratis', lo que genera gran confusión. Por ello a menudo en inglés se toman prestadas palabras españolas y se habla de 'libre software', en contraposición a 'gratis software', al igual que nosotros tomamos prestada la palabra *software*.

ANOTACIONES

Así pues no estamos hablando de software gratuito, y el software libre se puede vender si se desea. Pero debido a la tercera libertad, cualquiera puede redistribuirlo sin pedir dinero a cambio ni permiso a nadie, lo que hace prácticamente imposible obtener dinero por distribuirlo, salvo la pequeña cantidad que se pueda cargar por grabarlo en un soporte físico y enviarlo, algo raramente demandado excepto para grandes volúmenes, como es el caso de las *distribuciones*.

Se han formalizado definiciones más precisas de software libre, como es el caso notable de las directrices de la distribución Debian. En ellas se permite además que el autor exija que los códigos fuentes distribuidos no sean modificados directamente, sino que los originales se acompañen de parches separados y que se generen programas binarios con distinto nombre que el original. Además, exigen que las licencias no contaminen otros programas distribuidos en el mismo medio.

Lectura complementaria

[Ver bibliografía, apartado 85]

1.1.2. Términos relacionados

Equivalente a software libre es el término *Open Source Software* ('programas de fuente abierto'), promovido por Eric Raymond y la Open Source Initiative. Filosóficamente, el término es muy distinto, ya que hace énfasis en la disponibilidad de código fuente, no en la libertad, pero su definición es prácticamente la misma que la de Debian. Este nombre es más políticamente aséptico y recalca un aspecto técnico que puede dar lugar a ventajas técnicas, como mejores modelos de desarrollo y negocio, mayor seguridad, etc. Fuertemente criticado por Richard Stallman y la Free Software Foundation, ha encontrado mucho más eco en la literatura comercial y en las estrategias de las empresas que de una manera u otra apoyan el modelo.

Lectura complementaria

[Ver bibliografía, apartados 162, 183 y 23]

Otros términos relacionados de alguna manera con el software libre son:

- Freeware

Programas gratuitos. Normalmente se ceden en binario y con derechos de redistribución. Sin embargo, a veces sólo se pueden obtener

ANOTACIONES

de un sitio oficial, normalmente para promocionar otros programas o servicios, como es el caso de los kits de Java gratuitos que proporciona Sun Microsystems.

- Shareware

No es siquiera software gratis, sino un método de distribución, ya que los programas, generalmente sin fuentes, se pueden copiar libremente, pero no usar continuadamente sin pagarlos. La exigencia de pago puede estar incentivada por funcionalidad limitada o mensajes molestos, o una simple apelación a la moral del usuario, además de que las estipulaciones legales de la licencia podrían utilizarse en contra del infractor.

- Charityware, Careware

Generalmente shareware, pero cuyo pago se exige para una organización *caritativa* patrocinada. En muchos casos, el pago no se exige, pero se solicita una contribución voluntaria. Algún software libre, como vim solicita contribuciones voluntarias de este tipo.

Lectura complementaria

[Ver bibliografía, apartado 153]

- Dominio público

El autor renuncia absolutamente a todos sus derechos, en favor del común, lo cual tiene que estar declarado explícitamente en el programa, ya que si no se dice nada, el programa es propietario y no se puede hacer nada con él. En este caso, y si además se proporcionan los códigos fuentes, el programa es libre.

- Copyleft

Un caso particular de software libre cuya licencia obliga a que las modificaciones que se distribuyan sean también libres.

- Propietario, cerrado, no libre

Términos usados para denominar al software que no es libre ni de fuente abierta.

ANOTACIONES

1.2. Motivaciones

Como hemos visto hay dos grandes familias de motivaciones para el desarrollo de software libre, que dan lugar asimismo a los dos nombres con que se lo conoce:

- La motivación ética, abanderada por la Free Software Foundation, heredera de la cultura*hacker*, y partidaria del apelativo *libre*, que argumenta que el software es conocimiento y debe poderse difundir sin trabas. Su ocultación es una actitud antisocial y la posibilidad de modificar programas es una forma de libertad de expresión. Puede profundizarse en este aspecto en los ensayos de Stallman o en el análisis de Pekka Himanen.

Lectura complementaria

[Ver bibliografía, apartados 190 y 124]

- La motivación pragmática, abanderada por la Open Source Initiative y partidaria del apelativo *fuente abierta*, que argumenta ventajas técnicas y económicas, que repasaremos más adelante.

Lectura complementaria

[Ver bibliografía, apartado 43]

Aparte de estas dos grandes motivaciones, la gente que trabaja en software libre puede hacerlo por muchas otras razones, que van desde la diversión a la mera retribución económica, que es posible debido a *modelos de negocio* sustentables. En el capítulo 4 se profundiza en estas motivaciones a partir de análisis objetivos.

Lectura complementaria

[Ver bibliografía, apartado 195]

1.3. Consecuencias de la libertad del software

El software libre trae consigo numerosas ventajas y pocas desventajas, muchas de ellas exageradas (o falseadas) por la competencia propietaria. De ellas la que más fundamento tiene es la económica, ya que como vimos no es posible obtener mucho dinero de la distribución y ésta la puede y suele hacer alguien distinto al autor. Es por ello por lo que se necesitan modelos de negocio y otros mecanismos de financiación, que se desarrollan en el capítulo 5. Otras, como la falta de soporte o la calidad escasa, están relacionadas con la financiación, pero además en muchos casos son falsas, ya que incluso software sin ningún tipo de financiación suele ofrecer muy buen soporte a través foros de usuarios y desarrolladores, y muchas veces tiene gran calidad.

ANOTACIONES

Teniendo presentes los problemas económicos, hemos de observar que el modelo de costes del software libre es muy distinto del propietario, ya que gran parte de él se ha desarrollado fuera de la economía formal monetaria, muchas veces con mecanismos de trueque: "yo te doy un programa que te interesa y tú lo adaptas a tu arquitectura y le haces mejoras que a ti te interesan". En el capítulo 7 se explican mecanismos de ingeniería software apropiados para aprovechar estos recursos humanos no pagados y con características propias, mientras que en el capítulo 8 se estudian las herramientas usadas para hacer efectiva esta colaboración. Pero además gran parte de los costes disminuyen por el hecho de ser libre, ya que los programas nuevos no tienen por qué empezar desde cero, sino que pueden reutilizar software ya hecho. La distribución tiene también un coste mucho menor, ya que se hace vía Internet y con propaganda gratuita en foros públicos destinados a ello.

Otra consecuencia de las libertades es la calidad que se deriva de la colaboración voluntaria de gente que contribuye o que descubre y reporta errores en entornos y situaciones inimaginables por el desarrollador original. Además, si un programa no ofrece la calidad suficiente, la competencia puede tomarlo y mejorarlo, partiendo de lo que hay. Así, dos poderosos mecanismos: la *colaboración* y la *competencia* se combinan en aras de la calidad.

Examinemos ahora las consecuencias beneficiosas según el destinatario.

1.3.1. Para el usuario final

El usuario final, ya sea individual o empresa, puede encontrar verdadera competencia en un mercado con tendencia al monopolio. En particular, no depende necesariamente del soporte del fabricante del software, ya que puede haber múltiples empresas, quizá pequeñas, que disponiendo del código fuente y de conocimientos, puedan hacer negocio manteniendo determinados programas libres.

Ya no se depende tanto de la *fiabilidad* del fabricante para intentar deducir la calidad de un producto, sino que la guía nos dará la aceptación de la comunidad y la disponibilidad de los códigos fuentes. Nos olvidamos además de las *cajas negras*, en las que hay que con-

ANOTACIONES

fiar *porque sí*, y de las estrategias de los fabricantes, que pueden decidir unilateralmente dejar de mantener un producto.

La evaluación de productos antes de adoptarlos es ahora mucho más sencilla, ya que basta con instalar los productos alternativos en nuestro entorno real y probar, mientras que para el software propietario hay que fiarse de informes externos o negociar pruebas con los proveedores, lo cual no es siempre posible.

Dada la libertad de modificar el programa para uso propio, el usuario puede personalizarlo o adaptarlo a sus necesidades, corrigiendo errores si los tuviera. El proceso de corrección de errores descubiertos por los usuarios en software propietario suele ser extremadamente penoso, si no imposible, ya que si conseguimos que se repare, muchas veces se hará en la versión siguiente, que podría tardar años en salir, y a veces además habrá que adquirirla de nuevo. Sin embargo, lo podemos hacer nosotros si estamos cualificados o contratar el servicio fuera. También podemos, directamente o contratando servicios, integrar el programa con otro, o auditar su calidad (por ejemplo, la seguridad). El control pasa, en gran medida, del proveedor al usuario.

1.3.2. Para la Administración pública

La Administración pública es un gran usuario de características especiales, ya que tiene obligaciones especiales con el ciudadano, ya sea proporcionándole servicios accesibles, neutrales respecto a los fabricantes, ya garantizando la integridad, utilidad, privacidad y seguridad de sus datos a largo plazo. Todo ello la obliga a ser más respetuosa con los estándares que las empresas privadas y a mantener los datos en formatos abiertos y manipulados con software que no dependa de estrategia de empresas, generalmente extranjeras, certificado como seguro por auditoría interna. La adecuación a estándares es una característica notable del software libre que no es tan respetada por el software propietario, ávido en muchos casos notables de crear mercados cautivos.

Asimismo, la Administración tiene una cierta función de escaparate y guía de la industria que la hace tener un gran impacto, que debería dirigirse a la creación de un tejido tecnológico generador de

ANOTACIONES

riqueza nacional. Ésta puede crearse fomentando empresas, cuyo negocio sea en parte el desarrollo de nuevo software libre para la administración, o el mantenimiento, adaptación o auditoría del software existente. En el capítulo 6 nos extendemos más en esta cuestión.

1.3.3. Para el desarrollador

Para el desarrollador y productor de software, la libertad cambia mucho las reglas del juego. Con él le es más fácil competir siendo pequeño y adquirir tecnología punta. Puede aprovecharse del trabajo de los demás, compitiendo incluso con otro producto modificando su propio código, si bien también el competidor copiado se aprovechará de nuestro código (si es copyleft). Bien llevado, un proyecto puede conseguir la colaboración gratuita de mucha gente, del que el iniciado será la referencia. Asimismo, la distribución es barata y global. No obstante, como dijimos, el problema que tiene este desarrollador es la manera de obtener los recursos económicos si el trabajo no es de encargo. En el capítulo 5 veremos algo de esto.

1.3.4. Para el integrador

Para el integrador el software libre es el paraíso. No más cajas negras que intentar encajar, a menudo con ingeniería inversa. Puede limar asperezas e integrar trozos de programas para conseguir el producto integrado necesario, disponiendo de un acervo ingente de software libre de donde extraer las piezas.

1.3.5. Para el que proporciona mantenimiento y servicios

Disponer del código fuente lo cambia todo, situándonos casi en las mismas condiciones que el productor. Y no son las mismas porque hace falta un conocimiento profundo del programa que sólo el desarrollador posee, por lo que es conveniente que el mantenedor participe en los proyectos que se dedica a mantener. El valor añadido de los servicios es mucho más apreciado, ya que el coste del programa es bajo. Éste es actualmente el negocio más claro con software libre y con el que es posible un mayor grado de competencia.

ANOTACIONES

1.4. Otros recursos libres

¿Se pueden extender las ideas de los programas libres a otros recursos? Podemos pensar que otros recursos de información fácilmente copiables electrónicamente son de naturaleza similar a los programas, por lo que les son aplicables las mismas libertades, reglas y modelos de desarrollo y negocio. Hay diferencias cuyas implicaciones han hecho que no se desarrollen con la misma fuerza que los programas. La principal es que basta copiar los programas para que funcionen, mientras que desde que se copia otro tipo de información hasta que empieza a ser útil se ha de pasar por un proceso más o menos costoso, que puede ir desde el aprendizaje de un documento a la puesta en producción de un hardware descrito en un lenguaje apropiado. Revisemos los recursos libres más importantes.

1.4.1. Documentación de programas

La documentación que viene con un programa es parte integrante del mismo, como lo son los comentarios del código fuente. Así lo reconoce la Ley de Propiedad Intelectual, y parece lógico que se le apliquen las mismas libertades y evolucione de la misma manera. Toda modificación que se haga de un programa requiere un cambio simultáneo y consistente en su documentación.

La mayor parte de esta documentación suele estar codificada como ficheros de texto sin formato, ya que se pretende que sea universalmente accesible con un entorno de herramientas mínimo, y debe incluir una pequeña introducción al programa documentado (`README` o `LEEME`), instrucciones de instalación (`INSTALL`), alguna historia sobre la evolución pasada y futura del programa (`changelog` y `TODO`), autoría y condiciones de copia (`AUTHORS` y `copyright`), así como las instrucciones de uso. Todas ellas, menos la autoría y las condiciones de copia, deberían ser libremente modificables según el programa va evolucionando. A la autoría sólo se le deberían añadir nombres y créditos, pero sin borrar nada, y las condiciones de copia sólo deberían modificarse si estas mismas lo permiten, como veremos en el capítulo 3.

ANOTACIONES

Las instrucciones de uso suelen estar codificadas en formatos más complejos, ya que suelen ser documentos más largos y ricos. El software libre exige que esta documentación pueda ser modificada fácilmente, lo que a su vez obliga a usar formatos denominados *transparentes*, de especificación conocida y procesables por herramientas libres, como son, además del texto puro y limpio, el formato de páginas de manual de Unix, TexInfo, LaTeX o DocBook, sin perjuicio de distribuir también el resultado de transformar esos documentos fuente en formatos más aptos para visualizar o imprimir, como HTML, PDF o RTF (formatos *opacos*).

1.4.2. Otra documentación

Lectura complementaria

[Ver bibliografía, apartado 36]

Sin embargo, muchas veces se elabora documentación sobre programas por parte de terceros que no han intervenido en el desarrollo. A veces se trata de documentación de carácter didáctico que facilita la instalación y uso de un programa concreto (HOWTOs, COMOs o recetarios), a veces se trata de documentación más amplia, abarcando varios programas y su integración, comparando soluciones, etc., ya sea en forma tutorial o de referencia. En otras ocasiones, se trata de una mera recopilación de preguntas frecuentes con sus respuestas (FAQs o PUFs). Ejemplo notable es el *Proyecto de Documentación Linux*. En esta categoría podemos incluir otros documentos técnicos, no necesariamente sobre programas, ya sean las instrucciones para cablear una red local, construir una cocina solar, reparar un motor o seleccionar un proveedor de tornillos.

Estos documentos son algo intermedio entre mera documentación de programas y artículos o libros muy técnicos y prácticos. Sin menoscabo de la libertad de lectura, copia, modificación y redistribución, el autor puede querer verter opiniones que no desea que se tergiversen, o al menos que esas tergiversaciones no se le atribuyan. O puede querer que se conserven párrafos, como agradecimientos, o que necesariamente se modifiquen otros, como el título. Aunque estas inquietudes pueden también manifestarse con los programas en sí mismos, no se han mostrado con tanta fuerza en el mundo del software libre como en el de la documentación libre.

ANOTACIONES

1.4.3. Artículos científicos

El avance de la ciencia se debe en gran parte a que los investigadores que la hacen progresar para beneficio de la humanidad publican los resultados de sus trabajos en revistas de amplia difusión. Gracias a esa difusión los investigadores desarrollan un currículum que les permite progresar hacia puestos de mayor categoría y responsabilidad, a la vez que pueden obtener ingresos a partir de contratos de investigación que se han conseguido gracias al prestigio obtenido.

Así pues, esta difusión de artículos representa un *modelo de negocio* que se ha demostrado muy fructífero. Para que sea posible se necesita una amplia difusión y calidad garantizada. La difusión se ve obstaculizada por gran cantidad de revistas existentes, de coste no despreciable, cuya adquisición sólo es posible con presupuestos generosos. La calidad se garantiza por medio de la revisión por especialistas.

Por ello han surgido numerosas iniciativas de revistas en la Red, entre las que destacan la veterana *First Monday* o el proyecto *Public Library Of Science*. En "Directory of open access journals" se citan bastantes más. ¿Se debe permitir que personas que no son los autores publiquen una modificación de un artículo? Hay objeciones que alegan desde una posible falta de calidad o una tergiversación de opiniones o resultados, hasta el peligro de plagio fácil, que permite a algunos trepar sin esfuerzo y oscurecer los méritos de los verdaderos autores. Sin embargo, la obligación de citar al autor original y de pasar una revisión en una revista de prestigio puede contrarrestar esos problemas (ver apartado 3.3.2).

Lectura complementaria

[Ver bibliografía, apartados 22, 45 y 19]

Se ha querido establecer un paralelismo entre el software libre y la ciencia, ya que el modelo de desarrollo del primero implica la máxima difusión, la revisión por otros, presumiblemente expertos, y la reutilización de resultados.

Lectura complementaria

[Ver bibliografía, apartado 134]

1.4.4. Leyes y estándares

Hay documentos cuyo carácter es normativo, que definen cómo deben hacerse las cosas, ya sea para facilitar la convivencia entre las personas, ya para que programas o máquinas interoperen entre sí.

ANOTACIONES

Estos documentos requieren la máxima difusión, por lo que todo obstáculo a la misma es contraproducente. Por ello, es comprensible que tengan un tratamiento especial, como ocurre con la Ley de la Propiedad Intelectual española:



"No son objeto de propiedad intelectual las disposiciones legales o reglamentarias y sus correspondientes proyectos, las resoluciones de los órganos jurisdiccionales y los actos, acuerdos, deliberaciones y dictámenes de los organismos públicos, así como las traducciones oficiales de todos los textos anteriores."

La variante tecnológica de las leyes son las normas o estándares. En programación son especialmente importantes los protocolos de comunicaciones, ya sea entre máquinas remotas o entre módulos de la misma máquina. Es obvio que no debe limitarse su difusión, especialmente si queremos que florezcan los programas libres que interoperen con otros, pero a pesar de ello, tradicionalmente, los organismos de normalización, como la *ISO* e *ITU*, venden sus normas, incluso en formato electrónico, y prohíben su redistribución. Aunque pueda intentar justificarse esto para cubrir parcialmente los gastos, la libre difusión del texto de los estándares ha resultado mucho más productiva. Éste es el caso de las recomendaciones del *W3C* y sobre todo las que gobiernan Internet, disponibles desde el principio en documentos llamados *RFC*, de *Request for Comments*, en formatos electrónicos legibles en cualquier editor de textos.

Lectura complementaria

[Ver bibliografía, apartados 76 y 115]

Pero no es la disponibilidad la única causa del éxito de los protocolos de Internet. También lo es su *modelo de desarrollo*, muy similar al del software libre por su carácter abierto a la participación de cualquier interesado y por la utilización de listas de correo y medios similares.

Lectura complementaria

[Ver bibliografía, apartado 55]

¿Debe permitirse la modificación del texto de leyes y normas? Obviamente no si eso da lugar a confusión. Por ejemplo, sólo se admite que una RFC sea modificada para explicarla o añadirle comentarios aclaratorios, mientras que ni siquiera esto se permite sin autorización explícita para las recomendaciones del W3C. Las licencias son también documentos legales no modificables. ¿Debería permitirse la

ANOTACIONES

creación de nuevas normas derivadas de otras existentes a partir de los documentos originales? Probablemente hacerlo así llevaría a la proliferación fácil de normas similares e incompatibles que crearían confusión y podrían ayudar a empresas dominantes en el mercado a promover su propia variante incompatible, como de hecho ya está ocurriendo, especialmente en el ámbito de la web. No obstante, en el caso de las legislaciones de los estados, muchas veces se han copiado literalmente leyes de otros países, adaptadas con pequeñas modificaciones a las particularidades locales.

¿Existe un modelo de negocio para las leyes y normas? En torno a las leyes existen multitud de profesionales que se encargan de su diseño, interpretación y de forzar su aplicación (legisladores, abogados, procuradores, jueces, etc.). En torno a las normas existen laboratorios que otorgan certificados de conformidad. Las organizaciones de normalización viven, o deberían vivir, de las aportaciones de miembros interesados en promover estándares, porque, por ejemplo, su negocio está formado por productos que interoperan.

Lo mismo que es conveniente tener una definición de software libre o abierto, también es necesaria una definición de estándares abiertos. Bruce Perens ha propuesto una de ellas basada en los principios siguientes:

Lectura complementaria

[Ver bibliografía, apartado 14]

- Disponibilidad. Si es posible, proporcionar incluso una implementación libre de referencia.
- Maximizar las opciones del usuario final.
- Sin tasas sobre la implementación (no así sobre la certificación, aunque aconseja la disponibilidad de herramientas libres de *autocertificación*).
- Sin discriminación de implementador.
- Permiso de extensión o restricción (no certificable).
- Evitar prácticas predatorias por fabricantes dominantes. Toda extensión propietaria debe tener una implementación libre de referencia.

ANOTACIONES

1.4.5. Enciclopedias

Lectura complementaria

[Ver bibliografía, apartado 189]

En 1999 Richard Stallman lanza la idea de una enciclopedia libre como un mecanismo para evitar la apropiación del conocimiento y proporcionar acceso universal a documentación formativa. Estaría formada por artículos contribuidos por la comunidad, sin un control centralizado, donde distintos actores asumirían distintos roles, entre los que se aconseja, pero no se obliga, el de revisor. Esta enciclopedia no contendría solamente texto, sino elementos multimedios y software educativo libre.

Lectura complementaria

[Ver bibliografía, apartados 158 y 56]

Han surgido varias iniciativas para realizar esta visión. Por ejemplo, la *Nupedia* ha tratado, de momento sin éxito, de construir una enciclopedia de calidad, quizá por requerir un formato relativamente difícil de aprender (TEI), quizá por el requisito de que todos los artículos necesiten un editor, revisores científicos y de estilo, etc. Mucho más éxito ha tenido la *wikipedia*, que ha eliminado ambos obstáculos y se aproxima más a la idea de Stallman. La wikipedia usa una herramienta, el wiki, que permite a cualquiera editar cualquier documento por medio de un sistema de texto estructurado extraordinariamente simple, como veremos en el apartado 8.5.2. Debido a ello han podido recoger más de 120.000 artículos en sus dos primeros años de vida.

Lectura complementaria

[Ver bibliografía, apartado 206]

Cabe destacar también la *Concise Encyclopedia of Mathematics* con un concepto de libertad más limitado (sólo se puede consultar en la Red) y un modelo de desarrollo que necesariamente pasa todas las contribuciones por un comité editorial.

1.4.6. Cursos

Con la misma finalidad que las enciclopedias, se puede producir material docente libre, como apuntes, transparencias, ejercicios, libros, planificaciones o software didáctico. Existe una tendencia a ver las universidades como un negocio de producción y venta de conocimiento que contradice sus principios. Los motivos por los que una universidad puede poner a disposición de todo el mundo estos materiales son:

- Cumplir su misión como agente difusor del conocimiento.

ANOTACIONES

- No cuesta mucho hacer accesible a todo el mundo el material existente.
- Estos materiales no sustituyen a la enseñanza presencial.
- Es publicidad que puede atraer a alumnos y que contribuye al prestigio de la universidad.
- Permiten crear una comunidad de docentes que se revisen mutuamente los materiales y los mejoren.

La iniciativa más notable en este sentido es la del MIT, que prevé hacer accesibles más de 2.000 cursos de forma coherente, uniforme y bien catalogados.

Lectura complementaria

[Ver bibliografía, apartado 154]

1.4.7. Colecciones y bases de datos

La mera recolección de información siguiendo determinados criterios, ordenándola y facilitando su acceso es de por sí un producto de información valioso, independiente de la información en sí misma, sujeto por tanto a autoría y, por ello, a restricciones de las libertades de acceso, modificación y redistribución. Por tanto, si deseamos información libre también podemos desear colecciones libres.

Por ejemplo, podemos querer clasificar la información relevante en Internet, organizando y comentando enlaces. Esto es lo que hace ODP (*Open Directory Project*), operado por Netscape y mantenido por editores voluntarios organizados según un esquema jerárquico. El directorio completo puede copiarse libremente en formato RDF y publicarse modificado de alguna manera, como hacen Google y otros muchos buscadores que lo aprovechan. Netscape, propietario del directorio, garantiza un *contrato social* inspirado en el de la distribución Debian, que facilita la colaboración exterior, asegurando que siempre será libre, con políticas públicas, autorregulado por la comunidad, con los usuarios como primera prioridad.

Lectura complementaria

[Ver bibliografía, apartado 90, 42 y 87]

Otro ejemplo de colecciones interesantes para nosotros son las distribuciones de software libre, con los programas modificados para que encajen perfectamente entre sí y precompilados para su fácil ejecución.

ANOTACIONES

1.4.8. Hardware

Lectura complementaria

[Ver bibliografía, apartado 41]

La libertad en el hardware se puede analizar bajo dos aspectos. El primero es la necesidad de que las interfaces y juegos de instrucciones sean abiertos, de manera que cualquiera pueda realizar un manejador de dispositivo o un compilador para una arquitectura. El segundo es disponer de la información y el poder suficientes para poder reproducir un diseño hardware, modificarlo y combinarlo con otros. Los diseños pueden considerarse software en un lenguaje apropiado (VHDL, Verilog, etc.). Sin embargo, hacerlos funcionar no es fácil, ya que hay que fabricarlos, lo cual es caro y lento; si bien contamos con iniciativas en este sentido, entre las que podemos resaltar *Open Cores*, para circuitos integrados.

1.4.9. Literatura y arte

Para terminar nuestro recorrido por los recursos libres, no podemos olvidar el arte y la literatura, cuyo fin último no es tanto utilidad como estética. ¿Qué razones puede tener un artista para conceder libertades de copia, modificación y redistribución? Por un lado, dar a conocer al artista y favorecer la difusión de su obra, lo que puede permitirle obtener ingresos por otras actividades, como conciertos y obras de encargo. Por otro lado, favorecer la experimentación y la creatividad. En el arte ocurre lo mismo que en la técnica: la innovación es incremental y a veces es difícil distinguir el plagio de la pertenencia a un movimiento o corriente artística.

Lectura complementaria

[Ver bibliografía, apartado 24]

Obviamente no es lo mismo la creación que la interpretación, ni la música que la literatura. La música, la pintura, la fotografía y el cine son muy parecidos a los programas, en el sentido de que se les hace *funcionar* inmediatamente en un ordenador, mientras que otros, como la escultura, no pueden. No existen muchas iniciativas en arte y literatura libres, y éstas son muy diversas. Podemos mencionar las novelas del colectivo Wu Ming.

ANOTACIONES

2. Un poco de historia

Aunque todas las historias relacionadas con la informática son forzosamente breves, la del software libre es una de las más largas entre ellas. De hecho, podría decirse que el software nació libre y permaneció así durante su infancia. Sin embargo, con la llegada de la juventud, la situación cambió completamente. Sólo ahora, al llegar a su madurez, está en vías de recuperar la libertad. Esto no deja de ser curioso, pues para gran parte de los profesionales informáticos, el software propietario es el software *en su estado natural*. Afortunadamente, la situación es más bien la contraria, y las semillas del cambio que estamos empezando a entrever en los últimos años fueron plantadas ya a principios de la década de 1980.



Sugerencia

No hay muchas historias más o menos exhaustivas sobre el software libre, y las que existen son artículos más o menos limitados en el objeto de su estudio. En cualquier caso, el lector interesado puede completar lo expuesto en este capítulo consultando "Open Source Initiative. History of the osi" (énfasis en el impacto en el mundo empresarial, entre los años 1998 y 1999) o "Chris Rasch. A brief history of free/open source software movement", que cubre la historia del software libre hasta el año 2000, o "Nathan Newman. The origins and future of open source software, 1999", que se centra en gran medida en la promoción indirecta que el Gobierno de EE.UU. ha hecho del software libre, o sistemas similares, durante las décadas de 1970 y 1980.

Lectura recomendada

[Ver bibliografía, apartados 126, 169 y 157]

ANOTACIONES

2.1. El software libre antes del software libre

El software libre como concepto no apareció hasta principios de la década de 1980. Sin embargo, su historia puede trazarse desde bastantes años antes.

2.1.1. Y en el principio fue libre...

Durante los años 1960 el panorama de la informática estaba dominado por los grandes ordenadores, instalados fundamentalmente en empresas y centros gubernamentales. IBM era el principal fabricante, con gran diferencia sobre sus competidores. En esta época, cuando se adquiría un ordenador (el hardware), el software venía como un acompañante. Mientras se pagase el contrato de mantenimiento, se tenía acceso al catálogo de software que ofrecía el fabricante. Además, no era común la idea de que los programas fueran algo *separado* desde un punto de vista comercial.

Lectura complementaria

[Ver bibliografía, apartado 187]

En esta época el software se distribuía habitualmente junto con su código fuente (en muchos casos sólo como código fuente), y en general sin restricciones prácticas. Los grupos de usuarios como SHARE (usuarios de sistemas IBM) o DECUS (usuarios de DEC) participaban, y hasta cierto punto organizaban, estos intercambios. La sección *Algorithms* de la revista *Communications of the ACM* era otro buen ejemplo de foro de intercambio. Podría decirse que durante estos primeros años de la informática el software era libre, al menos en el sentido de que los que tenían acceso a él podían disponer habitualmente del código fuente, estaban acostumbrados a compartirlo, a modificarlo y a compartir las modificaciones. En palabras de Richard Stallman, refiriéndose a la situación en el Massachusetts Institute of Technology, uno de los lugares con más penetración de uso y más prestigio en investigación sobre informática en la época:



> "No denominábamos software libre a nuestro software porque este término no existía, pero eso es lo que era. Cuando alguien de otra universidad o de una empresa deseaba portar y usar un programa, se lo permitíamos

ANOTACIONES

> con gusto. Si veías a alguien usando un programa interesante y poco conocido, siempre podías pedir el código fuente para verlo, de manera que podías leerlo, cambiarlo, o canibalizar ciertas partes del mismo para hacer un nuevo programa."

El 30 de junio de 1969 IBM anunció que a comienzos de 1970 iba a empezar a vender parte de su software por separado. Esto supuso que sus clientes ya no podían obtener, incluido en el precio del hardware, los programas que necesitaban. El software se comenzó a percibir como algo con valor intrínseco y, como consecuencia, se hizo cada vez más habitual restringir escrupulosamente el acceso a los programas, y se limitaron, en la medida de lo posible (tanto técnica como legalmente), las posibilidades que tenían los usuarios para compartir, modificar o estudiar el software. En otras palabras, se pasó a la situación que aún hoy es la habitual en el mundo del software.

Lectura complementaria

[Ver bibliografía, apartado 119]



Sugerencia

El lector interesado en esta época de transición puede leer, por ejemplo, Larry Welke; Luanne Johnson (1998). *How the icp directory began*, donde Larry Welke comenta cómo nació uno de los primeros catálogos de software no ligados a un fabricante, y cómo en este proceso descubrió que las empresas estaban dispuestas a pagar por programas no creados por el fabricante de sus ordenadores.

Lectura recomendada

[Ver bibliografía, apartado 203]

A mediados de la década de 1970 era ya absolutamente habitual, en cualquier ámbito informático, encontrarse con software propietario. Esto supuso un gran cambio cultural entre los profesionales que trabajaban con software, a la vez que un gran número de empresas florecían en torno al nuevo negocio. Faltaba aún casi una década para que empezase a aparecer, de forma organizada y como reacción a esta situación, lo que hoy conocemos como software libre.

ANOTACIONES

2.1.2. Años 1970 y primeros 1980

Incluso cuando la tendencia abrumadoramente mayoritaria era la de explorar el modelo de software propietario, había iniciativas que mostraban algunas características de lo que luego se consideraría software libre. Alguna de ellas incluso produjo software libre, según la definición que usamos hoy día. Entre ellas, caben destacar Spice y TeX, además del caso mucho más complejo de Unix.

Spice (Simulation Program with Integrated Circuit Emphasis) es un programa desarrollado en la Universidad de California en Berkeley para simular las características eléctricas de un circuito integrado. Fue desarrollado y puesto en el dominio público por su autor, Donald O. Pederson, en 1973. SPICE era originalmente una herramienta docente, y como tal se extendió rápidamente a muchas universidades de todo el mundo, por lo que fue usado por muchos estudiantes de la que por aquel entonces era una incipiente disciplina de diseño de circuitos integrados. Estando en el dominio público, SPICE podía redistribuirse, modificarse, estudiarse, etc. Incluso se podía adaptar a unas necesidades concretas y vender esa versión como producto propietario (lo que se ha hecho durante su historia docenas de veces por parte de una gran cantidad de empresas). Con estas características, SPICE tenía todas las papeletas para convertirse en el estándar de la industria con sus diferentes versiones. Y efectivamente, eso fue lo que ocurrió. Probablemente éste fue el primer programa con características de software libre que durante un tiempo monopolizó un mercado, el de los simuladores de circuitos integrados, y sin duda pudo hacerlo precisamente por tener estas características (además de sus innegables cualidades técnicas).

Donald Knuth comenzó a desarrollar TeX durante un año sabático, en 1978. TeX es un sistema de tipografía electrónica muy utilizado para la producción de documentos de calidad. Desde el principio, Knuth utilizó una licencia que hoy sería considerada como de software libre. Cuando el sistema se consideró razonablemente estable, en 1985, mantuvo esa licencia. En esa época, TeX era un de los sistemas más grandes y más conocidos que podía considerarse software libre.

ANOTACIONES

2.1.3. Desarrollo temprano de Unix

Unix fue uno de los primeros sistemas operativos portables, creado originalmente por Thompson y Ritchie (entre otros) en los Bell Labs de AT&T. Su desarrollo ha continuado desde su nacimiento, hacia 1972, hasta hoy, con innumerables variantes comercializadas por literalmente decenas de empresas.

Durante los años 1973 y 1974, Unix llegó a muchas universidades y centros de investigación de todo el mundo, con una licencia que permitía su uso para fines académicos. Aunque había ciertas restricciones que impedían su distribución libre, entre las organizaciones que disponían de la licencia el funcionamiento fue muy similar al que se vio más tarde en muchas comunidades de software libre. Los que tenían acceso al fuente de Unix tuvieron un sistema que podían estudiar, mejorar y ampliar. Juntamente con él apareció una comunidad de desarrolladores, que pronto empezó a girar en torno al CSRG de la Universidad de California en Berkeley, y que desarrolló su propia cultura, que fue muy importante, como veremos más adelante, en la historia del software libre. Unix fue, hasta cierto punto, un ensayo temprano de lo que se vio con GNU y Linux varios años más tarde. Estaba confinado a una comunidad mucho más pequeña, y era necesaria la licencia de AT&T, pero en otros aspectos su desarrollo fue similar (en un mundo mucho menos comunicado).

Con el tiempo, Unix fue también un ejemplo temprano de los problemas que podían presentar los sistemas propietarios que a primera vista *tenían alguna característica del software libre*. Durante el final de la década de 1970, y sobre todo durante la de 1980, AT&T cambió su política, y el acceso a nuevas versiones de Unix se convirtió en algo difícil y caro. La filosofía de los primeros años, que hizo tan popular a Unix entre los desarrolladores, cambió radicalmente, hasta el punto de que en 1991 AT&T puso una demanda a la Universidad de Berkeley por publicar el código de Unix BSD que ellos (el CSRG de Berkeley) habían creado. Pero esa es otra historia, que retomaremos más adelante.

ANOTACIONES

2.2. El comienzo: BSD, GNU

Todos los escenarios descritos hasta ahora son, o bien iniciativas individuales, o bien no cumplen los requisitos del software libre. En cualquier caso, hasta principios de la década de 1980 no aparecieron, de forma organizada y consciente, los primeros proyectos para la creación de sistemas compuestos de software libre, y lo que probablemente es más importante: los fundamentos éticos, legales y hasta económicos, que luego se continuarían desarrollando hasta el día de hoy. De esta época procede también el propio término *software libre*.

2.2.1. Richard Stallman, GNU, FSF: nace el movimiento del software libre

A principios de 1984, Richard Stallman, en aquella época empleado en el AI Lab del MIT, abandonó su trabajo para comenzar el proyecto GNU. Stallman se consideraba un *hacker* de los que gozaban compartiendo sus inquietudes tecnológicas y su código. Veía con desagrado cómo su negativa a firmar acuerdos de exclusividad y no compartición le estaban convirtiendo en un extraño en su propio mundo, y cómo el uso de software propietario en su entorno le dejaba impotente antes situaciones que antes podía solventar fácilmente.

Lectura complementaria

[Ver bibliografía, apartado 187]

Su idea al abandonar el MIT era construir un sistema de software completo, de propósito general, pero completamente libre. El sistema (y el proyecto que se encargaría de hacerlo realidad) se llamó GNU (acrónimo recursivo, *GNU's Not Unix*). Aunque desde el principio el proyecto GNU incluyó en su sistema software ya disponible (como Tex, o más adelante, el sistema X Window), había mucho que construir. Richard Stallman comenzó por escribir un compilador de C (GCC) y un editor (Emacs), ambos aún en uso (y muy populares) hoy día.

Lectura complementaria

[Ver bibliografía, apartado 96]

Desde el principio del proyecto GNU, Richard Stallman estaba preocupado por las libertades que tendrían los usuarios de su software. Estaba interesado en que no sólo los que recibieran los programas directamente del proyecto GNU, sino cualquiera que lo recibiera después de cualquier número de redistribuciones y (quizás) modificaciones, siguiera disfrutando de los mismos derechos (modificación,

ANOTACIONES

redistribución, etc.). Para ello, escribió la licencia GPL, probablemente la primera licencia de software diseñada específicamente para garantizar que un programa fuera libre en este sentido. Al mecanismo genérico que utilizan las licencias tipo GPL para conseguir estas garantías, Richard Stallman lo llamó *copyleft*, que hoy día es el nombre de una gran familia de licencias de software libre.

Richard Stallman también fundó la Free Software Foundation (FSF con el fin de conseguir fondos para el desarrollo y la protección del software libre, y sentó los fundamentos éticos del software libre, con documentos como *The GNU Manifesto* y *Why Software Should Not Have Owners*.

Lectura complementaria

[Ver bibliografía, apartados 95 y 186]

Desde el punto de vista técnico, el proyecto GNU fue concebido como un trabajo muy estructurado y con metas muy claras. El método habitual estaba basado en grupos relativamente pequeños de personas (habitualmente voluntarios) que desarrollaban alguna de las herramientas que luego encajarían perfectamente en el rompecabezas completo (el sistema GNU). La modularidad de UNIX, en la que se inspiraba el desarrollo, encajaba perfectamente en esta idea. El método de trabajo generalmente implicaba el uso de Internet, pero ante la escasa implantación de aquellos días, la Free Software Foundation también vendía cintas en las que grababa las aplicaciones, siendo probablemente uno de las primeras organizaciones en beneficiarse económicamente (aunque de manera bastante limitada) de la creación de software libre.

A principios de la década de 1990, unos seis años después de su nacimiento, el proyecto GNU estaba muy cerca de tener un sistema completo similar a Unix. Aun así, hasta ese momento todavía no había producido una de las piezas fundamentales: el kernel del sistema (el núcleo del sistema operativo que se relaciona con el hardware y permite que todo funcione). Sin embargo, el software de GNU era muy popular entre los usuarios de las distintas variantes de Unix, por aquella época el sistema operativo más usado en las empresas. Además, el proyecto GNU había conseguido ser relativamente conocido entre los profesionales informáticos, y muy especialmente entre los que trabajaban en universidades. En esa época, sus productos ya gozaban de una merecida reputación de estabilidad y calidad.

ANOTACIONES

ANOTACIONES

2.2.2. El CSRG de Berkeley

El CSRG (Computer Science Research Group) de la Universidad de California en Berkeley fue, desde 1973, uno de los centros donde más se desarrolló todo lo relacionado con Unix durante los años 1979 y 1980. No sólo se portaron aplicaciones y se construyeron otras nuevas para su funcionamiento sobre Unix, sino que se hicieron importantes mejoras al kernel, y se le añadió mucha funcionalidad. Por ejemplo, durante la década de los años 1980, varios contratos de DARPA (dependiente del Ministerio de Defensa de EE.UU.) financiaron la implementación de TCP/IP que ha sido considerada hasta nuestros días como la referencia de los protocolos que hacen funcionar Internet (vinculando, de paso, el desarrollo de Internet y la expansión de las estaciones de trabajo con Unix). Muchas empresas utilizaron como base de sus versiones de Unix los desarrollos del CSRG, dando lugar a sistemas muy conocidos en la época, como SunOS (Sun Microsystems) o Ultrix (Digital Equipment). De esta manera, Berkeley se convirtió en una de las dos fuentes fundamentales de Unix, junto con la *oficial*, AT&T.

Para poder utilizar todo el código que producía el CSRG (y el de los colaboradores de la comunidad Unix que ellos de alguna manera coordinaban), hacía falta la licencia de Unix de AT&T, que cada vez era más difícil (y más cara) de conseguir, sobre todo si se quería el acceso al código fuente del sistema. Tratando de evitar en parte este problema, en junio de 1989 el CSRG liberó la parte de Unix relacionada con TCP/IP (la implementación de los protocolos en el kernel y las utilidades), que no incluía código de AT&T. Fue la llamada *Networking Release 1* (Net-1). La licencia con que se liberó fue la famosa *licencia BSD* que, salvo ciertos problemas con sus cláusulas sobre obligaciones de anuncio, ha sido considerada siempre como un ejemplo de licencia libre minimalista (que además de permitir la redistribución libre, también permitía su incorporación a productos propietarios). Además, el CSRG probó un novedoso método de financiación (que ya estaba probando con éxito la FSF): vendía cintas con su distribución por 1.000 dólares. Aunque cualquiera podía a su vez redistribuir el contenido de las cintas sin ningún problema (la licencia lo permitía), el CSRG vendió cintas a miles de organizaciones, con lo que consiguió fondos para seguir desarrollando.

Dado el éxito de la distribución de Net-1, Keith Bostic propuso reescribir todo el código que aún quedaba del Unix original de AT&T. A pesar del escepticismo de algunos miembros del CSRG, realizó un anuncio público pidiendo ayuda para realizar esta tarea, y poco a poco las utilidades reescritas a partir de especificaciones comenzaron a llegar a Berkeley. Mientras tanto, el mismo proceso se realizó con el kernel, y se reescribió de forma independiente la mayor parte del código que no se había realizado por parte de Berkeley o por colaboradores voluntarios. En junio de 1991, y después de conseguir el permiso de la Administración de la Universidad de Berkeley, se distribuyó la *Networking Release 2* (Net-2), con casi todo el código del kernel y todas las utilidades de un sistema Unix completo. De nuevo, todo esto se distribuyó bajo la licencia BSD, y de nuevo se vendieron miles de cintas al precio de 1.000 dólares la unidad.

Sólo seis meses después de la liberación de Net-2, Bill Jolitz escribió el código que faltaba en el kernel para que funcionase sobre arquitectura i386, liberando 386BSD, que fue distribuida por Internet. A partir de este código surgieron, en sucesión, todos los sistemas de la familia *BSD. Primero apareció NetBSD, como una recopilación de los parches que se habían ido portando en la Red para mejorar 386BSD. Más adelante apareció FreeBSD como un intento de soportar fundamentalmente la arquitectura i386. Varios años más tarde se formó el proyecto OpenBSD, con énfasis en la seguridad. Y también hubo una distribución propietaria a partir de Net-2 (aunque era ciertamente original, ya que ofrecía a sus clientes todo el código fuente como parte de la distribución básica), realizada de forma independiente por BSDI.

En parte como reacción a la distribución hecha por BSDI, Unix System Laboratories (USL), subsidiaria de AT&T que tenía los derechos de la licencia de Unix, puso una demanda judicial, primero a BSDI y luego a la Universidad de California. En ella les acusaba de distribuir su propiedad intelectual sin permiso. Después de varias maniobras judiciales (incluyendo una contrademanda de la Universidad de California contra USL), Novell compró los derechos de Unix a USL, y en enero de 1994 llegó a un acuerdo extrajudicial con la Universidad de California. Como resultado de este acuerdo, el CSRG distribuyó la versión 4.4BSD-Lite, que fue pron-

ANOTACIONES

to utilizada por todos los proyectos de la familia *BSD. Poco después (tras liberar aún la versión 4.4BSD-Lite Release 2), el CSRG desapareció. En ese momento hubo quien temió que ello representara el fin de los sistemas *BSD, pero el tiempo ha demostrado que siguen vivos y coleando, con una nueva forma de gestión, más típicamente de proyectos libres. Hoy día, los proyectos que gestionan la familia *BSD son de los más antiguos y consolidados en el mundo del software libre.



Sugerencia

La historia del desarrollo de Unix BSD, desde sus orígenes en una cinta que llevó Bob Fabry a Berkeley con una de las primeras versiones del código de Thompson y Ritchie para hacerla funcionar en un PDP-11 que compraron conjuntamente los departamentos de informática, estadística y matemáticas, hasta las demandas judiciales de AT&T y las últimas liberaciones de código que dieron lugar a la familia de sistemas operativos libres *BSD es muy ilustrativa de una cierta forma de desarrollar software durante los años 1970 y 1980. Quien esté interesado en ella puede disfrutar de la lectura de Marshall Kirk McKusick. “Twenty Years of Berkeley Unix. From AT&T-Owned to Freely Redistributable”. En: DiBona et al. (1999), donde se comentan muchos detalles de esta época.

Lectura recomendada

[Ver bibliografía, apartado 149]

2.2.3. Los comienzos de Internet

Casi desde su nacimiento, a principios de la década de 1970, Internet tuvo mucha relación con el software libre. Por un lado, desde sus comienzos, la comunidad de desarrolladores que la construyeron tuvieron claros varios principios que luego se harían clásicos en el mundo del software libre. Por ejemplo, la importancia de dar posibilidades a los usuarios para que ayuden a depurar los errores, o la compartición del código de las implementaciones. La importancia de BSD Unix en su desarrollo (al proporcionar durante los años 1980 la implementación más popular de los protocolos TCP/IP) hizo que mu-

ANOTACIONES

chas costumbres y formas de funcionamiento pasasen fácilmente de una comunidad (la de desarrolladores alrededor del CSRG) a otra (los que estaban construyendo NSFNet, y luego simplemente Internet), y viceversa. Muchas de las aplicaciones básicas en el desarrollo de Internet, como Sendmail (servidor de correo) o Bind (implementación del servicio de nombres) fueron libres, y en gran medida fruto de esta colaboración entre comunidades.

Por último, la comunidad del software libre, durante el final de los años 1980 y la década de 1990, fue una de las primeras en explorar hasta el fondo las nuevas posibilidades que permitía Internet para la colaboración entre grupos geográficamente dispersos. Esta exploración fue la que hizo posible, en gran medida, la propia existencia de la comunidad BSD, la FSF o el desarrollo de GNU/Linux.

Uno de los aspectos más interesantes del desarrollo de Internet, desde el punto de vista del software libre, fue la gestión completamente abierta de sus documentos y sus normas. Aunque hoy pueda parecer algo normal (pues es lo habitual, por ejemplo, en el IETF o el WWW Consortium), en su época la libre disposición de todas las especificaciones y documentos de diseño, incluyendo las normas que definen los protocolos, fue algo revolucionario y fundamental para su desarrollo. Podemos leer en Michael Hauben and Ronda Hauben. (1997). *Netizens. On the History and Impact of Usenet and the Internet* (pág. 106). IEEE Computer Society Press.



"Este proceso abierto promovía y llevaba al intercambio de información. El desarrollo técnico tiene éxito sólo cuando se permite que la información fluya libre y fácilmente entre las partes interesadas. La promoción de la participación es el principio fundamental que hizo posible el desarrollo de la Red."

Observe el lector cómo este párrafo podría ser suscrito, casi con toda seguridad, por cualquier desarrollador refiriéndose al proyecto de software libre en el que colabora.

Lectura complementaria

[Ver bibliografía, apartado 118]

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 174]

ANOTACIONES

En otra cita, Lawrence Roberts. "The evolution of packet switching". *Proceedings of the IEEE* (vol.66, pág. 267) podemos leer:



> "Como ARPANET era un proyecto público que conectaba muchas de las principales universidades e instituciones de investigación, los detalles de implementación y rendimiento se publicaban ampliamente."

Obviamente, esto es lo que ocurre hoy día en los proyectos de software libre, donde toda la información relacionada con el proyecto (y no sólo la implementación) suele ser pública.

En este ambiente, y hasta que Internet ya bien entrados los años 1990 se convirtió sobre todo en un negocio, la comunidad de usuarios y su relación con los desarrolladores era clave. En esa época muchas organizaciones aprendieron a confiar no en una única empresa proveedora del servicio de comunicación de datos, sino en una compleja combinación de empresas de servicios, fabricantes de equipos, desarrolladores profesionales y voluntarios, etc. Las mejores implementaciones de muchos programas no eran las que venían con el sistema operativo que se compraba con el hardware, sino implementaciones libres que rápidamente las sustituían. Los desarrollos más innovadores eran fruto no de grandes planes de investigación en empresas, sino de estudiantes o profesionales probando ideas y recogiendo la realimentación que les enviaban cientos de usuarios que usaban sus programas libres.

Como ya se ha dicho, Internet también proporcionó al software libre las herramientas básicas para colaborar a distancia. El correo electrónico, los grupos de News, los servicios de FTP anónimo (que fueron los primeros almacenes masivos de software libre), y más tarde los sistemas de desarrollo integrados basados en web han sido fundamentales (e imprescindibles) para el desarrollo de la comunidad del software libre tal como la conocemos, y en particular para el funcionamiento de la inmensa mayoría de los proyectos de software libre. Desde el principio, proyectos como GNU o BSD hicieron un uso masivo e intenso de todos estos mecanismos, desarrollando, a la vez

que las usaban, nuevas herramientas y sistemas que a su vez mejoraban Internet.



Sugerencia

El lector interesado en una historia de la evolución de Internet, escrita por varios de sus protagonistas, puede consultar Barry M. Leiner; Vinton G. Cerf; Robert E. Kahn; David D. Clark; Leonard Kleinrock; Daniel C. Lynch; Jon Postel; Larry G. Roberts; Stephen Wolff (1997). *A brief history of the intert. Communications of the ACM*.

Lectura recomendada

[Ver bibliografía, apartado 145]

2.2.4. Otros proyectos

Durante la década de 1980 vieron la luz otros importantes proyectos libres. Entre ellos destaca, por su importancia y proyección futura, el sistema X Window (sistema de ventanas para sistemas tipo Unix), desarrollado en el MIT, y que fue uno de los primeros ejemplos de financiación en gran escala de proyectos libres con recursos de un consorcio de empresas. También merece la pena mencionar Ghostscript, un sistema de gestión de documentos Postscript desarrollado por una empresa, Aladdin Software, que fue uno de los primeros casos de búsqueda de modelo de negocio produciendo software libre.

En general, y sobre todo a finales de los años 1980, están en marcha toda una serie de pequeños (y no tan pequeños) proyectos libres. Todos ellos, junto con los grandes proyectos mencionados hasta aquí, estaban ya sentando las bases de los primeros sistemas libres completos que aparecieron a principios de la década de 1990.

2.3. Todo en marcha

Hacia 1990, gran parte de los componentes de un sistema informático completo estaban ya listos como software libre. Por un lado, el proyecto GNU y por otro, las distribuciones BSD habían completado

ANOTACIONES

la mayor parte de las aplicaciones que componen un sistema operativo. Asimismo, proyectos como X Window o el propio GNU habían construido desde entornos de ventanas hasta compiladores, que en muchos estaban entre los mejores de su género (por ejemplo, muchos administradores de sistemas SunOS o Ultrix sustituían para sus usuarios las aplicaciones propietarias de su sistema por las versiones libres de GNU o de BSD). Para tener un sistema completo construido sólo con software libre, faltaba únicamente un componente: el kernel. Dos esfuerzos separados e independientes vinieron a rellenar este hueco: 386BSD y Linux.

2.3.1. En busca de un kernel

A finales de la década de los ochenta, principios de los noventa, el proyecto GNU contaba con una gama básica de utilidades y herramientas que permitían tener el sistema operativo al completo. Ya entonces muchas aplicaciones libres eran las mejores en su campo (utilidades UNIX, compiladores), siendo especialmente interesante el caso de X Window. Sin embargo, para terminar el rompecabezas faltaba una pieza esencial: el núcleo del sistema operativo. El proyecto GNU estaba buscando esa pieza con un proyecto llamado Hurd, que pretendía construir un kernel con tecnologías modernas.

2.3.2. La familia *BSD

Prácticamente en la misma época, la comunidad BSD estaba también en camino hacia un kernel libre. En la distribución Net-2 sólo faltaban 6 ficheros para tenerlo (el resto ya había sido construido por el CSRG o sus colaboradores). A principios de 1992 Bill Jolitz completa esos ficheros y distribuye 386BSD, un sistema que funciona sobre arquitectura i386, y que con el tiempo dará lugar a los proyectos NetBSD, FreeBSD y OpenBSD. El desarrollo durante los meses siguientes es rápido, y a finales de año ya es suficientemente estable como para ser usado en producción en entornos no críticos, incluyendo, por ejemplo, un entorno de ventanas gracias al proyecto XFree (que había portado X Window a la arquitectura i386) o un compilador de gran calidad, GCC. Aunque hay componentes que usaban otras licencias (como los procedentes del proyecto GNU, que

ANOTACIONES

usaban la GPL), la mayor parte del sistema se distribuye bajo la licencia BSD.



Sugerencia

Algunos de los episodios de esta época son ilustrativos de la potencia de los modelos de desarrollo de software libre. El caso de Linus Torvalds, desarrollando Linux mientras era estudiante de segundo curso de la Universidad de Helsinki es bien conocido. Pero no es el único caso de un estudiante que se abrió camino gracias a sus desarrollos libres. Por ejemplo, Thomas Roel, un estudiante alemán, portó X11R4 (una versión del sistema X Window) a un PC basado en un 386. Este desarrollo le llevó a trabajar en Dell, y más adelante a ser fundador de los proyectos X386 y XFree, básicos para que GNU/Linux y los *BSD tuvieran pronto un entorno de ventanas. Puede leerse más sobre la historia de XFree, y el papel de Roel en ella, en Michael J. Hammel. "The history of xfree86". *Linux Magazine* (diciembre, 1991).

Lectura recomendada

[Ver bibliografía, apartado 114]

Luego vino la demanda de USL, que propició que muchos potenciales usuarios temieran ser a su vez demandados si la Universidad de California perdía el juicio, o simplemente que el proyecto se parara. Quizás ésta fue la razón de que más adelante la base instalada de GNU/Linux fuera mucho mayor que la de *BSD combinados. Pero esta cuestión es algo difícil de asegurar.

2.3.3. GNU/Linux entra en escena

En julio de 1991 Linus Torvalds (estudiante finés de 21 años) pone el primer mensaje donde menciona su (por entonces) proyecto de hacer un sistema libre similar a Minix. En septiembre libera la primerísima versión (0.01), y cada pocas semanas aparecieron nuevas versiones. En marzo de 1994 apareció la versión 1.0, la primera que fue denominada *estable*, pero el kernel que había construido Linus era usable desde bastantes meses antes. Durante este periodo, lite-

ralmente cientos de desarrolladores se vuelcan sobre Linux, integrando a su alrededor todo el software de GNU, XFree, y muchos otros programas libres. A diferencia de los *BSD, Linux (el kernel) y gran parte de los componentes que se integran alrededor de él se distribuyen con la licencia GPL.

Lectura recomendada

[Ver bibliografía, apartado 117]



Sugerencia

La historia de Linux es probablemente una de las más interesantes (y conocidas) en el mundo del software libre. Quien esté interesado en ella puede encontrar muchos enlaces a información sobre la misma en las páginas del décimo aniversario de su anuncio, http://www.linux10.org/history/, aunque probablemente la más interesante es Ragib Hasan. *History of Linux*. Como curiosidad, puede consultarse el hilo en que Linus Torvalds anunciaba que estaba empezando a crear lo que luego fue Linux (en el grupo de News comp.os.minix) en http://groups.google.comgroups?th=d161e94858c4c0b9. En él explica cómo lleva trabajando en su kernel desde abril, y cómo ya ha portado algunas herramientas del proyecto GNU sobre éste (concretamente, menciona bash y gcc).

Entre los muchos desarrollos aparecidos en torno a Linux, uno de los más interesantes es el concepto de *distribución*. Los primeros aparecieron ya en 1992 (MCC Interim Linux, de la Universidad de Manchester, TAMU, de Texas A&M, y la más conocida SLS, que más tarde dio lugar a Slackware, que aún se distribuye hoy día), y han supuesto la entrada de la competencia en el mundo del empaquetamiento de sistemas alrededor de Linux. Cada distribución trata de ofrecer a sus usuarios objetivo un GNU/Linux listo para usar, y basándose todas en el mismo software, han de competir en mejoras que su base de usuarios considere importantes. Además de proporcionar paquetes precompilados y listos para usar, las distribuciones suelen ofrecer sus propias herramientas para gestionar la selección, instalación, sustitución y desinstalación de estos paquetes, la instalación inicial en un ordenador, y la gestión y administración del sistema operativo.

ANOTACIONES

Con el tiempo, unas distribuciones han ido sucediéndose a otras como las más populares. Entre todas ellas, cabe destacar algunas:

- Debian
- RedHat
- SuSE
- Mandrake

2.4. Tiempos excitantes

Desde que GNU/Linux tiene una presencia habitual en los medios, desde que la inmensa mayoría de empresas utiliza software libre al menos para algunos de sus procesos informáticos, y desde que es difícil ser un estudiante de informática y no utilizar software libre en grandes cantidades, está claro que todo esto ya no es un asunto de unos cuantos "enteradillos", sino que se está convirtiendo en algo muy importante para el sector. En el fondo, por su puesto, late una pregunta muy importante: ¿Estamos ante un nuevo modelo de industria software? Porque todo esto del software libre podría ser una moda pasajera, que con el tiempo sólo será recordada con nostalgia, o un nuevo modelo que está aquí para quedarse, y quizás para cambiar radicalmente una de las industrias más jóvenes, pero también de las más influyentes.

2.4.1. Finales de 1990

A mediados de la década de 1990 el software libre ofrece ya entornos completos (distribuciones de GNU/Linux, sistemas *BSD) que permiten el trabajo diario de mucha gente, sobre todo de desarrolladores de software. Aún hay muchas asignaturas pendientes (la mayor de ellas, el disponer de mejores interfaces gráficas de usuario, en una época donde Windows 95 es considerado el estándar), pero también unos cuantos miles de personas en todo el mundo ya sólo usan software libre en su trabajo diario. Los anuncios de nuevos pro-

ANOTACIONES

yectos se suceden y el software libre comienza su largo camino de expansión hacia las empresas, los medios de comunicación y en general el conocimiento público.

De esta época es también el despegue de Internet como red para todos, en muchos casos de la mano de programas libres (sobre todo en su infraestructura). La llegada de la web a los hogares de millones de usuarios finales consolida esta situación, al menos en lo que se refiere a servidores: los servidores web (HTTP) más populares siempre han sido libres (primero el servidor del NCSA, luego Apache).

Lectura complementaria

[Ver bibliografía, apartados 171, y 120]

Quizás el comienzo del camino del software libre hasta su puesta de largo en la sociedad pueda situarse en el célebre ensayo de Eric Raymond, "La catedral y el bazar". Aunque mucho de lo expuesto en él era ya bien conocido por la comunidad de desarrolladores de software libre, el hecho de escribirlo en un artículo y darle una difusión grande fuera de la comunidad *tradicional* lo convirtió en una influyente herramienta de promoción del concepto de software libre como mecanismo de desarrollo, alternativo al que usaba la industria del software tradicional. Otro artículo muy importante de esta época fue "Setting Up Shop: The Business of Open-Source Software", de Frank Hecker, que por primera vez expuso los modelos de negocio posibles en torno al software libre, y que fue escrito para influir en la decisión sobre la liberación del código del navegador de Netscape.

De hecho, si el artículo de Raymond supuso una gran herramienta de difusión de algunas de las características fundamentales del software libre, la liberación del código del navegador de Netscape supuso el primer caso en que una empresa relativamente grande, de un sector muy innovador (la entonces naciente industria del web) tomaba la decisión de liberar como software libre uno de sus productos. En aquella época, el Netscape Navigator estaba perdiendo la batalla de los navegadores frente al producto de Microsoft (Internet Explorer), en parte por las tácticas de Microsoft de combinarlo con su sistema operativo. Para muchos, Netscape hizo lo único que podía hacer: tratar de cambiar las reglas para poder competir con un gigante. Y de este cambio de reglas (tratar de competir con un modelo de software libre) nació el proyecto Mozilla. Este proyecto, no sin problemas, ha llevado varios años después a un navegador que si bien no ha recuperado una parte significativa de la enorme cuota de mer-

ANOTACIONES

cado que tuvo en su día el Netscape Navigator, parece que técnicamente es al menos tan bueno como sus competidores propietarios (incluido el propio Internet Explorer).

En cualquier caso, y con independencia de su éxito posterior, el anuncio de Netscape de liberar el código de su Navigator supuso un fuerte impacto en la industria del software. Muchas empresas comenzaron a considerar el software libre como algo digno de estudio, y algo que podría interesarles mucho.

También los mercados financieros se empezaron a ocupar del software libre. En plena euforia de las puntocom, varias empresas de software libre se convierten en objetivo de inversores. Quizás el caso más conocido es el de Red Hat, una de las primeras empresas que reconocieron que la venta de CD con sistemas GNU/Linux listos para usar podía ser un modelo de negocio. Red Hat comenzó distribuyendo su Red Hat Linux, con gran énfasis (al menos para lo habitual en la época) en la facilidad de manejo y mantenimiento del sistema por personas sin conocimientos específicos de informática. Con el tiempo, Red Hat fue diversificando su negocio, manteniéndose en general en la órbita del software libre, y en septiembre de 1998 anunció que Intel y Netscape habían invertido en ella. *Si es bueno para Intel y Netscape, seguro que es bueno para nosotros*, debieron de pensar muchos inversores. Y así, cuando Red Hat salió a bolsa en el verano de 1999, la oferta pública de acciones fue suscrita completamente, y pronto el valor de cada título subió como la espuma. Fue la primera vez que una empresa consiguió financiación del mercado de valores con un modelo basado en el software libre. Pero no fue la única: pronto le siguieron otras como VA Linux o Andover.net.



Sugerencia

Red Hat proporciona una lista de hitos históricos relacionados con su empresa con motivo de su décimo aniversario (1993-2003) en http://www.redhat.com/mktg/rh10year/.

En esta época nacen también muchas empresas basadas en modelos de negocio en torno al software libre. Sin salir a bolsa ni lograr

ANOTACIONES

estupendas capitalizaciones, han sido sin embargo muy importantes para el desarrollo del software libre. Por ejemplo, aparecieron muchas otras empresas que empezaron distribuyendo sus propias versiones de GNU/Linux, como SuSE (Alemania), Conectiva (Brasil) o Mandrake (Francia). Otras proporcionan servicios a empresas que ya demandan mantenimiento y adaptación de productos libres: LinuxCare (EE.UU.), Alcove (Francia), ID Pro (Alemania), y muchas más.

Por su parte, los gigantes del sector también empiezan a posicionarse ante el software libre. Algunas empresas, como IBM, lo incorporan directamente en su estrategia. Otras, como Sun Microsystems, mantienen con él una curiosa relación, a veces de apoyo, a veces de indiferencia, a veces de enfrentamiento. La mayoría (como Apple, Oracle, HP, SGI, etc.) exploran el modelo del software libre con diversas estrategias, que van desde la liberación selectiva de software hasta el simple porte a Linux de sus productos, pasando por todo un espectro de actuaciones, como utilización más o menos intensiva de software libre en sus productos, o exploración de modelos de negocio basados en el servicio de soporte a productos libres.

Desde el punto de vista técnico, lo más destacable de esta época es, probablemente, la aparición de dos ambiciosos proyectos con el objetivo de conseguir llevar el software libre al entorno de escritorio (*desktop*) de los usuarios no muy versados en la informática: KDE y GNOME. El objetivo final era, dicho de forma muy simplista, que no hubiera que usar la línea de órdenes para interaccionar con GNU/Linux o *BSD, ni con los programas sobre esos entornos.

KDE fue anunciado en octubre de 1996. Utilizando las bibliotecas gráficas Qt (por aquel entonces un producto propietario de la empresa Troll Tech, pero gratuito para su uso sobre Linux), iniciaron la construcción de un conjunto de aplicaciones de escritorio que funcionasen de forma integrada, y tuvieran un aspecto uniforme. En julio de 1998 liberaron la versión 1.0 del K Desktop Environment, que pronto fue seguida de nuevas versiones cada vez más completas y maduras. Las distribuciones de GNU/Linux pronto incorporaron KDE como escritorio para sus usuarios (o al menos como uno de los entornos de escritorios que sus usuarios podían elegir).

ANOTACIONES

En gran medida como reacción a la dependencia que tenía KDE de la biblioteca propietaria Qt, en agosto de 1997 se anuncia el proyecto GNOME, con objetivos y características muy similares a las de KDE, pero con el objetivo explícito de que todos sus componentes sean libres. En marzo de 1999 se liberó GNOME 1.0, que también se iría, con el tiempo, mejorando y estabilizando. A partir de ese momento, la mayor parte de las distribuciones de sistemas operativos libres (y muchos derivados de Unix propietarios) ofrecieron como opción el escritorio de GNOME o el de KDE, y las aplicaciones de ambos entornos.

Lectura complementaria

[Ver bibliografía, apartado 82]

Simultáneamente, los principales proyectos de software libre que ya estaban en marcha continúan con buena salud, y surgen nuevos proyectos cada día. En varios nichos de mercado se observa cómo la mejor solución (reconocida por casi todo el mundo) es software libre. Por ejemplo, Apache ha mantenido casi desde su aparición en abril de 1995 la mayor cuota de mercado entre los servidores web. XFree86, el proyecto libre que desarrolla X Window, es con diferencia la versión de X Window más popular (y por tanto, el sistemas de ventanas para sistemas tipo Unix más extendido). GCC es reconocido como el compilador de C más portable, y uno de los de mejor calidad. GNAT, sistema de compilación para Ada 95, se hace con la mayor parte del mercado de compiladores Ada en pocos años. Y así sucesivamente...

En 1998 se creó la Open Source Initiative (OSI), que decidió adoptar el término *open source software* ('software de fuente abierta') como una marca para introducir el software libre en el mundo comercial, tratando de evitar la ambigüedad que en inglés supone el término *free* (que significa tanto 'libre' como 'gratis'). Esta decisión supuso (y aún supone) uno de los debates más enconados del mundo del software libre, ya que la Free Software Foundation y otros consideraron que era mucho más apropiado hablar de *software libre*. En cualquier caso, la OSI realizó una fructífera campaña de difusión de su nueva marca, que ha sido adoptada por muchos como la forma preferida de hablar del software libre, sobre todo en el mundo anglosajón. La OSI utilizó para definir el software *open source* una definición derivada de la que utiliza el proyecto Debian para definir qué es software libre (que por otra parte refleja con bastante aproximación la idea de la FSF al respecto), por lo que desde el punto de vista práctico, casi cualquier programa que es considerado software libre es también

Lectura complementaria

[Ver bibliografía, apartados 185, 85 y 98]

ANOTACIONES

considerado *open source*, y viceversa. Sin embargo, las comunidades del software libre y del software de fuente abierta (o al menos las personas que se identifican como parte de una o de otra) pueden ser profundamente diferentes.

2.4.2. Principios de los 2000

A principios de la década del 2000 el software libre es un serio competidor en el segmento de servidores, y comienza a estar ya listo para el escritorio. Sistemas como GNOME 2.x, KDE 3.x y OpenOffice pueden ser utilizados por usuarios domésticos, y son suficientes para las necesidades de muchas empresas, al menos en lo que a ofimática se refiere. Los sistemas libres (y fundamentalmente los basados en Linux) son fáciles de instalar, y la complejidad para mantenerlos y actualizarlos es comparable a la de otros sistemas propietarios (incluyendo Windows).

En estos momentos, cualquier empresa de la industria del software tiene una estrategia con respecto al software libre. La mayoría de las grandes multinacionales (IBM, HP, Sun, Corel, Apple, Oracle) incorpora el software libre con mayor o menor decisión. En un extremo podríamos situar a empresas como Oracle, por ejemplo, que reaccionan simplemente portando sus productos a GNU/Linux. En el otro podría situarse a IBM, que tiene la estrategia más decidida y ha realizado las mayores campañas de publicidad sobre Linux. Entre los líderes del mercado informático, sólo Microsoft se ha significado con una estrategia claramente contraria al software libre, y en particular al software distribuido bajo licencia GPL.

En cuanto al mundo del software libre en sí mismo, a pesar de los debates que de vez en cuando sacuden la comunidad, el crecimiento es enorme. Cada vez hay más desarrolladores, más proyectos de software libre activos, más usuarios, etc. Cada vez más el software libre está pasando de ser algo marginal para convertirse en un competidor a tener en cuenta.

Ante este desarrollo, aparecen nuevas disciplinas que estudian específicamente el software libre, como la ingeniería del software libre. A partir de sus investigaciones, comenzamos poco a poco a entender cómo funciona en sus diferentes aspectos: modelos de desarrollo, de

ANOTACIONES

negocio, mecanismos de coordinación, gestión de proyectos libres, motivación de desarrolladores, etc.

En estos años comienzan también a verse los primeros efectos de la *deslocalización* que permite el desarrollo de software libre: países considerados como *periféricos* participan en el mundo del software libre de forma muy activa. Por ejemplo, es significativo el número de desarrolladores mexicanos o españoles (ambos países con poca tradición de industria software) en proyectos como GNOME [lancashire:code-culture-cash]. Y, por supuesto, es aún más interesante el papel de Brasil, con empresas como Conectiva, una gran cantidad de desarrolladores y expertos en tecnologías de software libre, y un decidido apoyo por parte de las administraciones públicas. Mención aparte merece el caso de LinEx, muy significativo de cómo una región con poca tradición de desarrollo de software puede tratar de cambiar la situación con una estrategia agresiva de implantación de software libre.

Desde el punto de vista de la toma de decisiones a la hora de implantar soluciones software, es de destacar cómo hay ciertos mercados (como los servicios de Internet, o la ofimática) donde el software libre se ha convertido en una opción natural, y es difícil de justificar el hecho de no considerarla cuando se está estudiando qué tipo de sistema utilizar.

Como aspecto negativo, estos años han visto de qué modo el entorno legal donde se mueve el software libre está cambiando rápidamente en todo el mundo. Por un lado, las patentes de software (patentes de programación) están siendo consideradas cada vez en más países. Por otro, las nuevas leyes de protección de derechos de autor están dificultando o haciendo imposible el desarrollo de aplicaciones libres en algunos ámbitos, siendo el más conocido el de los visores de DVD (debido al algoritmo CSS de ofuscación de imágenes que se utiliza en esa tecnología).

LinEx

A principios del 2002 la Junta de Extremadura dio a conocer públicamente el proyecto LinEx. La idea es simple: promover la creación de una distribución basada en GNU/Linux con el objetivo fundamen-

ANOTACIONES

tal de utilizarla en los miles de ordenadores que va a instalar en los centros educativos públicos de toda la región. Extremadura, situada en la parte occidental de España, fronteriza con Portugal, cuenta con aproximadamente un millón de habitantes, y nunca se había destacado por sus iniciativas tecnológicas. De hecho, la región prácticamente carece de industria de software.

En este contexto, LinEx ha supuesto una aportación muy interesante en el panorama del software libre a escala mundial. Mucho más allá de ser una nueva distribución de GNU/Linux basada en Debian (lo que no deja de ser algo relativamente anecdótico), y más allá de su enorme impacto en medios de comunicación (es la primera vez que Extremadura ha salido en portada del Washington Post, y una de las primeras que lo ha hecho un producto de software libre, por ejemplo), lo extraordinario es la (al menos aparentemente) sólida apuesta de una administración pública por el software libre. La Junta de Extremadura ha decidido probar un modelo diferente en la enseñanza de informática, y a medio plazo probablemente en el propio uso de la informática dentro de sus competencias. Este hecho la convierte en la primera administración pública de un país desarrollado que toma decididamente este camino. A raíz de la iniciativa de la Junta ya está empezando a producirse movimiento, tanto dentro como fuera de Extremadura. Hay academias que enseñan informática con LinEx; se han escrito libros para apoyar esta enseñanza; hay ordenadores que se venden con LinEx preinstalado. En general, se está tratando de crear en torno a esta experiencia todo un tejido docente y empresarial que le proporcione soporte. Asimismo, se está empezando a exportar la experiencia: la Junta de Andalucía (en el sur de España, 8 millones de habitantes) ya ha anunciado GuadaLinex, una iniciativa hasta cierto punto similar.

Knoppix

Desde finales de los años 1990 existen distribuciones de GNU/Linux que se instalan fácilmente, pero probablemente Knoppix, cuya primera versión apareció durante el 2002, ha llevado este concepto a su máxima expresión. Un CD que arranca prácticamente en cualquier PC, convirtiéndolo (sin tener siquiera que formatear el disco, ya que permite su uso *en vivo*) en una máquina GNU/Linux completamente funcional, con una selección de las herramientas más habi-

ANOTACIONES

tuales. Knoppix une una buena detección automática de hardware con una buena selección de programas y un funcionamiento ”en vivo”. Permite, por ejemplo, una experiencia rápida y directa de lo que puede suponer trabajar con GNU/Linux. Y está dando lugar a toda una familia de distribuciones del mismo tipo, especializadas en necesidades específicas de un perfil concreto de usuarios.

OpenOffice.org

En 1999 Sun Microsystems compró una empresa alemana llamada StarDivision, cuyo producto estrella era StarOffice, un juego de herramientas ofimático similar en funcionalidad a Office, el juego de herramientas de Microsoft. Un año más tarde, Sun distribuyó gran parte del código de StarOffice bajo una licencia libre (la GPL), dando lugar al proyecto OpenOffice.org. Este proyecto liberó la versión 1.0 de OpenOffice.org en mayo del 2002. OpenOffice.org se ha convertido en un juego de aplicaciones ofimáticas de calidad y funcionalidad similar a la de cualquier otro producto ofimático, y lo que es más importante, interopera muy bien con los formatos de datos de MS Office. Estas características han hecho de ella la aplicación de referencia del software libre en el mundo de la ofimática.

La importancia de OpenOffice.org, desde el punto de vista de extensión del software libre a un gran número de usuarios, es enorme. Por fin, ya es posible cambiar, prácticamente sin traumas, de los entornos propietarios habituales en ofimática (sin duda la aplicación estrella en el mundo empresarial) a entornos completamente libres (por ejemplo, GNU/Linux más Gnome y/o KDE más OpenOffice.org).

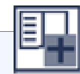
Mozilla, Galeon y los demás

Prácticamente desde su aparición en 1994 hasta 1996, Netscape Navigator fue el líder indiscutible del mercado de navegadores web, con cuotas de mercado de hasta el 80%. La situación empezó a cambiar cuando Microsoft incluyó el Internet Explorer con su Windows 95, lo que supuso que poco a poco fuera perdiendo cuota de mercado. A principios de 1998 Netscape anunció que iba a distribuir gran parte del código de su Navigator como software libre, cosa que efecti-

ANOTACIONES

vamente hizo en marzo del mismo año lanzando el proyecto Mozilla. Durante bastante tiempo estuvo rodeado de incertidumbre, e incluso pesimismo (por ejemplo, cuando el líder del proyecto, Jamie Zawinski, lo abandonó), dado que pasaba el tiempo y no aparecía ningún producto como resultado de su lanzamiento.

En enero del 2000, el proyecto liberó Mozilla M13, que fue considerada como la primera versión razonablemente estable. Pero sólo en mayo del 2002 se publicó finalmente la versión 1.0, la primera oficialmente estable, más de cuatro años después de la liberación del primer código del Navigator.



Sugerencia

En Brian Wilson. *Netscape navigator* puede consultarse una reseña detallada de las principales versiones de Netscape Navigator y Mozilla, así como de sus principales características.

Lectura recomendada

[Ver bibliografía, apartado 211]

Por fin Mozilla era una realidad. Aunque quizás demasiado tarde, teniendo en cuenta las cuotas de mercado que presenta durante el periodo 2002-2003 Internet Explorer (líder indiscutible del mercado, que ha relegado a Mozilla y a otros a una posición completamente marginal). Pero el proyecto Mozilla, a pesar de tardar tanto, ha dado sus frutos. No sólo los que cabía esperar (el navegador Mozilla), sino muchos otros que podrían considerarse *colaterales*, como por ejemplo Galeon, otro navegador basado en el mismo motor de HTML.

Mozilla ha ayudado a completar un gran hueco en el mundo del software libre. Antes de la aparición de Konqueror (el navegador del proyecto KDE), no había muchos navegadores libres con interfaz gráfica. A partir de la publicación de Mozilla, han ido apareciendo gran cantidad de proyectos basados en él, produciendo una buena cantidad de navegadores. Por otro lado, la combinación Mozilla más OpenOffice permite usar software libre para las tareas más cotidianas incluso en un entorno Windows (ambos funcionan no sólo sobre GNU/Linux, *BSD y otros sistemas tipo Unix, sino que también lo hacen sobre Windows). Esto permite, por primera vez en la historia del software libre, que la transición de software propietario a libre en en-

ANOTACIONES

tornos de oficina sea simple: se puede empezar usando estas dos aplicaciones sobre Windows, sin cambiar de sistema operativo (en el caso de los que lo usan habitualmente), y con el tiempo eliminar la única pieza no libre pasando a GNU/Linux o FreeBSD.

2.5. El futuro: ¿una carrera de obstáculos?

Sin duda, es difícil predecir el futuro. Y desde luego, no es algo a lo que nos vayamos a dedicar aquí. Por lo tanto, más que tratar de explicar cómo será el futuro del software libre, trataremos de mostrar los problemas que previsiblemente tendrá que afrontar (y de hecho lleva ya tiempo afrontando). De cómo sea el mundo del software libre capaz de superar estos obstáculos dependerá, sin duda, su situación dentro de unos años.

- Técnica FUD (*fear, uncertainity, doubt*, o en español, 'miedo', 'desconocimiento', 'duda'). Son técnicas bastante habituales en el mundo de las tecnologías de la información, y que hasta hoy han sido utilizadas por los competidores de productos de software libre para tratar de desacreditarlos, con mayor o menor razón, y con éxito variable. En líneas generales, el software libre, quizás debido a su complejidad y diversos métodos de penetración en las empresas, ha resultado bastante inmune a estas técnicas.

- Disolución. Muchas empresas están probando los límites del software libre como modelo, y en particular tratando de ofrecer a sus clientes modelos que presentan algunas características similares al software libre. El principal problema que puede presentar este tipo de modelos es la confusión que generan en los clientes y desarrolladores, que tienen que estudiar con mucho detalle la letra pequeña para darse cuenta de que lo que se les está ofreciendo no tiene las ventajas que para ellos supone el software libre. El caso más conocido de modelos de este tipo es el programa Shared Source de Microsoft.

- Desconocimiento. En muchos casos los usuarios llegan al software libre simplemente porque creen que es gratis. O porque lo consideran *de moda*. Si no profundizan más allá, y estudian con

ANOTACIONES

cierto detenimiento las ventajas que les puede ofrecer el software libre como modelo, corren el riesgo de no aprovecharse de ellas. En muchos casos, las suposiciones de partida en el mundo del software libre son tan diferentes de las habituales en el mundo del software propietario que es indispensable un mínimo análisis para comprender que lo que en un caso es habitual, en el otro puede ser imposible, y viceversa. El desconocimiento, por lo tanto, no puede sino generar insatisfacciones y pérdida de oportunidades en cualquier persona y organización que se aproxime al software libre.

- **Impedimentos legales**. Sin duda éste es el principal problema con el que se va a encontrar el software libre en los próximos años. Aunque el entorno legal donde se desarrolló el software libre durante la década de 1980 y la primera mitad de la de 1990 no era ideal, al menos dejaba suficiente espacio para que creciese en libertad. Desde entonces, la extensión del ámbito de la patentabilidad al software (que se ha producido en muchos países desarrollados) y las nuevas legislaciones sobre derechos de autor, que limitan la libertad de creación del desarrollador de software, suponen cada vez barreras más altas a la entrada del software libre en segmentos importantes de aplicaciones.

2.6. Algunas fechas de la historia del software libre

Lectura complementaria

[Ver bibliografía, apartados 111 y 126]

Ésta es sólo una lista de fechas que podrían ser consideradas como importantes en la historia del software libre. Está basada en la que aparece en *Working group on Libre Software. Free software / open source: Information society opportunities for europe?* (1999) y en ella ofrece la Open Source Initiative, que no pretende ser completa: es seguro que muchas fechas importantes no aparecen. Sin embargo, probablemente ofrece una visión lo suficientemente completa del panorama histórico de la evolución del mundo del software libre.

Décadas de 1950 y 1960:

El software se distribuye con su código fuente y sin restricciones en grupos de usuarios como SHARE (IBM) y DECUS (DEC).

ANOTACIONES

1969, abril:

Se publica la RFC número 1, que describe la primera Internet (entonces ARPANET). La libre disposición de las RFC, y en particular de las especificaciones de los protocolos usados en Internet fue un factor clave de su desarrollo.

1970, enero:

IBM comienza a vender su software por separado, dando lugar al comienzo de la industria del software propietario.

1972:

Unix comienza a distribuirse en universidades y centros de investigación.

1973:

Unix llega a la Universidad de California en Berkeley. Comienza la historia de Unix BSD.

1973:

SPICE es puesto por Donald O. Pederson en el dominio público. Con el tiempo se convertirá en la referencia en su campo (simuladores de circuitos integrados).

1978:

Donald Knuth, de la Universidad de Stanford, comienza a trabajar en TeX, un sistema de composición electrónica que se distribuyó como software libre.

1983:

Richard Stallman escribe el Manifiesto de GNU, donde pide la vuelta a la compartición pública de software.

1984:

Comienza el proyecto GNU. Los desarrolladores que colaboran en él, inicialmente coordinados por Richard Stallman, comienzan a crear un gran número de herramientas similares a las que había en Unix, incluyendo un editor (Emacs) y un compilador (GCC). La meta es construir un sistema operativo completamente libre.

ANOTACIONES

1985:

El X Consortium, basado en el MIT, distribuye el sistema X Window como software libre, bajo una licencia muy poco restrictiva.

1985:

Richard Stallman funda la Free Software Foundation. Entre otros fines, funcionará como centro receptor de fondos y recursos que ayuden al desarrollo del proyecto GNU, y como dueño de la propiedad intelectual generada por el proyecto.

1989:

Se funda Cygnus, la primera empresa dedicada fundamentalmente a proporcionar servicios comerciales para el software libre (entre ellos, soporte, desarrollo y adaptación de programas libres).

1990:

La Free Software Foundation anuncia su intento de construir un kernel que se llamará GNU Hurd. La meta de este proyecto es completar el mayor hueco que queda en la estrategia del proyecto GNU de construir un sistema operativo completo.

1991:

William y Lynne Jolitz escriben una serie en Dr. Dobbs Journal sobre cómo portar BSD Unix a PC basados en i386.

1991, agosto:

Linus Torvalds, estudiante finés de informática con 21 años, anuncia que ha empezado a trabajar en un kernel tipo Unix libre, usando herramientas de GNU, como GCC. Su meta en esa época es construir un *Minix Libre*.

1991, octubre:

Linus Torvalds libera la primera versión de su kernel, aún muy primitiva, que es llamada Linux.

1992:

El Ejército del Aire de EE.UU. concede un contrato a la Universidad de Nueva York para construir un compilador libre para la nueva versión de Ada (en la época, lenguaje de uso casi obligado en los contratos con el ejército de EE.UU.), Ada 95. El equipo de

ANOTACIONES

NYU elige a GNU GCC para la generación de código, y llama a su compilador GNAT (GNU NYU Ada 95 Translator).

1992, julio:

William y Lynne Jolitz liberan 386BSD 0.1, que con el tiempo dará lugar a los proyectos NetBSD, FreeBSD, y más tarde, OpenBSD.

1993:

Se funda SuSE, en Alemania, que comienza sus negocios distribuyendo Slackware Linux, traducida al alemán.

1993, agosto:

Ian Murdock comienza una nueva distribución basada en Linux, llamada Debian GNU/Linux. Se convertirá en la distribución construida por desarrolladores voluntarios con más participantes.

1993, diciembre:

FreeBSD 1.0, una de las primeras distribuciones estables descendientes del 386BSD de los Jolitz, es liberada en Internet.

1994:

Se funda la empresas Ada Core Technologies, por los desarrolladores de GNAT, con el objetivo de asegurar su desarrollo y evolución futuras, y con un modelo de negocio basado en ofrecer servicios en torno al compilador y a sus clientes (y no en vender el compilador, que sigue siendo software libre). Con el tiempo, GNAT se convertirá en el líder del mercado de compiladores Ada.

1994, enero:

Se libera la versión 0.91 de Debian GNU/Linux, fruto del esfuerzo de 12 desarrolladores.

1994:

Marc Ewing comienza la distribución Red Hat Linux. Igual que era el caso de Debian, trata de mejorar los resultados de la distribución dominante en la época, Slackware.

1994, marzo:

Se publica el primer número de Linux Journal.

ANOTACIONES

1994, octubre:

Liberación de NetBSD 1.0.

1995:

Bob Young funda Red Hat Software, comprando la distribución Red Hat Linux a su autor Marc Ewing y fusionándola con su propio negocio, ACC, que vendía materiales relacionados con Linux y Unix, por catálogo, desde 1993. Poco después se publica Red Hat Linux 2.0, la primera en incluir el formato de paquetes RPM.

1995, enero:

Liberación de FreeBSD 2.0.

1995, abril:

Primera liberación oficial de Apache (0.6.2)

1996:

First Conference on Freely Redistributable Software (primer congreso sobre software redistribuible libremente). Cambridge, Massachusetts, USA.

1996, octubre:

Anuncio del proyecto KDE, uno de los primeros en atacar los problemas de usabilidad en entorno Unix, y el primero que trata de hacerlo en gran escala en el mundo del software libre.

1997, enero:

Eric S. Raymond presenta su artículo "The Cathedral and the Bazaar" ('La catedral y el bazar'), con sus opiniones sobre por qué funcionan ciertos modelos de desarrollo del software libre.

1997, agosto:

Miguel de Icaza anuncia el proyecto GNOME, un *competidor* de KDE con metas similares, pero con el objetivo explícito que todo el sistema resultante fuera software libre. Nació como reacción de la Free Software Foundation y otros a los problemas de licencias que tenía KDE, donde un componente fundamental, la biblioteca Qt, no era software libre en aquella época.

ANOTACIONES

1998, 22 de enero:

Netscape declara su intención de distribuir como software libre el código de su Navigator, que había sido el líder en el mercado de navegadores de web.

1998, 3 de febrero:

Chris Peterson, Todd Anderson, John Hall, Larry Augustin, Sam Ockman y Eric Raimond se reúnen para estudiar las consecuencias del anuncio de Netscape de liberar su navegador, y deciden promover el término *open source software* (software de fuente abierta), usándolo como una marca que garantice que los productos que la llevan son software libre. Los promotores de este término entienden que es más adecuado para el mundo empresarial que el habitual hasta ese momento, *free software* ('software libre'). Para gestionar el término, se crea la Open Source Initiative.

1998, 31 de marzo:

Netscape publica en Internet gran parte del código fuente de su Navigator.

1998, 7 de mayo:

Corel anuncia el Netwinder, un *network computer* basado en Linux. Es la primera vez que una gran empresa comercializa un aparato cuyo software es básicamente software libre. Poco después Corel anuncia planes para portar su software de ofimática (entre el que se encuentra WordPerfect) a Linux, lo que también es novedoso en la época.

1998, 28 de mayo:

Sun Microsystems y Adaptec entran a formar parte de Linux Internacional. Son las primeras grandes empresas informáticas que lo hacen.

1998, junio:

La conferencia técnica de USENIX, tradicionalmente dedicada a Unix, abre una sesión paralela llamada FREENIX, centrada en el software libre.

ANOTACIONES

1998, 22 de junio:

IBM anuncia que comercializará y proporcionará soporte para Apache, utilizándolo como el servidor de su línea de productos WebSphere.

1998, julio:

Se libera Debian GNU/Linux 2.0, construida por más de 300 voluntarios. Esta distribución incluía más de 1.500 paquetes.

1998, julio:

Se libera KDE 1.0, la primera versión distribuida como *estable*. Varias distribuciones de GNU/Linux la incorporan poco después.

1998, agosto:

Linus Torvalds y Linux aparecen en la portada de la revista Forbes.

1998, 29 de septiembre:

Red Hat, por entonces la empresa líder en el mercado de distribuciones basadas en Linux, anuncia que Intel y Netscape han adquirido una participación minoritaria en su capital. El software libre comienza a despertar el interés de los inversores.

1998, noviembre:

Se funda MandrakeSoft, que publica poco después Mandrake Linux, su distribución de GNU/Linux.

1998, 1 de noviembre:

Publicación de los *documentos de Halloween*, en los que supuestamente Microsoft identifica a GNU/Linux y el software libre como un competidor importante, y planea cómo combatirle.

1999, 27 de enero:

HP y SGI anuncian que soportarán Linux en sus máquinas, lo que marca el comienzo de una tendencia: el abandono de los Unix propietarios por los fabricantes de ordenadores que los usaban como *su* sistema operativo, en favor de Linux.

ANOTACIONES

1999, marzo:

Se libera GNOME 1.0, que fue más adelante estabilizada (October GNOME), siendo incorporada en varias distribuciones de GNU/Linux.

1999, 15 de marzo:

Apple libera Darwin, que será el componente central de su nuevo MacOSX, bajo una licencia libre.

1999, agosto:

Red Hat sale a bolsa. El precio de las acciones sube enormemente en los primeros días tras la salida, y llega a estar capitalizada en 4.800 millones de dólares. Más adelante saldrían a bolsa otras empresas relacionadas con el software libre, como VA Linux y Andover.net. Todas estas empresas bajarán su cotización extraordinariamente unos años más tarde, cuando explota la burbuja de las puntocom.

1999, octubre:

Se fundan dos empresas para producir software en el marco del proyecto GNOME: Eazel (que quebró en el 2002, después de producir Nautilus, un gestor de ficheros para GNOME) y Helix Code (más adelante renombrada como Ximian, que ha producido, entre otras, herramientas como Red Carpet o Evolution).

1999, noviembre:

Red Hat Software compra Cygnus. La empresa resultante es la más grande del mundo en el campo del software libre.

2000, enero:

Publicación de Mozilla M13, considerada como la primera versión razonablemente estable de Mozilla, casi dos años después de la liberación de gran parte del código de Netscape Navigator.

2000, mayo:

Se libera GNOME 1.2 (Bongo GNOME).

2000, agosto:

Se anuncia la creación de la Fundación GNOME.

ANOTACIONES

ANOTACIONES

2001, enero:

Se publica la versión 2.4 de Linux.

2002, 3 de abril:

Se publica KDE 3.0, la tercera generación del entorno de escritorio KDE. Los escritorios libres empiezan a estar a la altura de los escritorios comerciales tradicionales.

2002, abril:

Anuncio público del proyecto LinEx, con el que la Junta de Extremadura (España) pretende utilizar su propia distribución de GNU/Linux para informatizar los colegios públicos de la región.

2002, mayo:

Publicación de Mozilla 1.0, la primera versión oficialmente estable del proyecto.

2002, mayo:

Publicación de OpenOffice.org 1.0. Pronto esta aplicación se convertirá en la referencia ofimática en el mundo del software libre.

2002, 26 de junio:

Se publica GNOME 2.0, que supone un gran paso adelante de cara al usuario con una interfaz más cuidada y atención a la usabilidad. También se introducen aspectos para mejorar la accesiblidad.

2002, 28 de julio:

Se publica la versión 3.0 de Knoppix, una distribución de evaluación que permite ser instalada en el disco duro de manera sencilla y rápida. Knoppix se convierte en un fulgurante éxito.

2002, diciembre:

Red Hat Software anuncia que ha tenido flujo de caja positivo los trimestres segundo y tercero de 2002.

2003, enero:

MandrakeSoft, empresa productora de la distribución Mandrake Linux, se declara en suspensión de pagos.

2003, 28 de mayo:

El Ayuntamiento de Munich (Alemania) anuncia que reemplazará Windows con Linux en la mayor parte de su sistema informático.

2003, julio:

MandrakeSoft anuncia que ha tenido flujo de caja positivo durante todo el año, y que espera salir de su situación de suspensión de pagos a finales del 2003.

2003, 7 de julio:

Carta abierta a la Organización Mundial de la Propiedad Intelectual (WIPO, World Intellectual Proprierty Organization) para que examine los nuevos modelos de creación colectiva abierta (entre los que se encuentra el software libre, pero también el proyecto Genoma Humano o las revistas científicas abiertas).

Lectura complementaria

[Ver bibliografía, apartado 197]

2003, 15 de julio:

Se funda la Mozilla Foundation. Netscape Inc. (propiedad de AOL) anuncia que dejará de desarrollar el navegador Netscape y, por tanto, su tutela del proyecto Mozilla. La Fundación Mozilla se constituye con una donación de dos millones de dólares por parte de AOL y apoyo material y humano de varias empresas, entre ellas la propia AOL, Red Hat y Sun Microsystems.

2003, 4 de agosto:

Novell compra Ximian Inc., una de las empresas líderes en el desarrollo de software libre –en especial para GNOME–, como parte de su estrategia para asentarse en el mercado de soluciones sobre Linux.

ANOTACIONES

3. Aspectos legales

En este capítulo se presentan los principales aspectos legales relacionados con el software libre. Para ponerlos en contexto, se empieza por una pequeña introducción a los conceptos más básicos de la propiedad intelectual e industrial, antes de exponer la definición detallada de software libre, software de fuente abierta y otros conceptos relacionados. Se analizan también con cierto detalle las licencias de software libre más habituales y su impacto sobre los modelos de negocio (que se tratará con más detalle en el capítulo 5) y los modelos de desarrollo. También se comentarán licencias para otros recursos libres distintos del software.

3.1. Breve introducción a la propiedad intelectual

Con el término *propiedad intelectual* se agrupan distintos privilegios que se otorgan sobre bienes intangibles con valor económico. De ellos podemos destacar los de *copyright* (derechos de autor) y similares, que protegen de la copia no autorizada los trabajos literarios o artísticos, programas de ordenador, recopilaciones de datos, diseños industriales, etc.; las marcas, que protegen símbolos; las indicaciones geográficas, que protegen denominaciones de origen; el secreto industrial, que respalda la ocultación de información, y las patentes, que otorgan monopolio temporal sobre un invento a cambio de desvelarlo. En muchas legislaciones, entre ellas la española, se distingue la *propiedad intelectual*, que se refiere a los derechos de autor, de la *propiedad industrial*, que abarca las figuras restantes. En el ámbito internacional, la OMPI o WIPO (Organización Mundial de la Propiedad Intelectual, según siglas en español o en inglés) promueve ambos tipos de propiedad en todos sus aspectos, mientras que el acuerdo TRIPS (*aspectos comerciales de la propiedad intelectual*) establece unos mínimos de protección y obliga a todos los países miembros de la OMC o WTO (Organización Mundial del Comercio) a desarrollarlos en unos plazos que dependen del nivel de desarrollo del país.

ANOTACIONES

Aunque la Declaración Universal de los Derechos Humanos (art. 27) reconoce a las personas el derecho a que se protejan los intereses morales y materiales que le correspondan por razón de las producciones científicas, literarias o artísticas de que sean autores, en la práctica este derecho suele ser transferido a las empresas que emplean a los creadores o que comercializan sus creaciones. No obstante, y a pesar de ello, la propiedad intelectual se justifica no sólo por razones morales, sino prácticas, para dar cumplimiento a otro derecho: el de la sociedad a beneficiarse de las creaciones, incentivándolas con beneficios y protegiendo las inversiones para la creación, investigación y desarrollo. Para armonizar ambos derechos, la propiedad intelectual es temporal, caducando cuando ha cumplido su función de promoción.

Pero la caducidad no es la única característica que diferencia la propiedad intelectual de la ordinaria. Los objetos de la misma pueden copiarse fácil y económicamente, sin pérdida de calidad, La copia no perjudica a quien ya disfruta de lo copiado, al contrario que el robo, que sí que priva del objeto al poseedor original. La copia sí que puede perjudicar al propietario, ya que le priva potencialmente de los ingresos de una venta. El control de la copia de intangibles es mucho más complicado que el del robo de bienes tangibles y puede llevarnos a una sociedad policial, que necesite controlar todas las copias de información, y a una gran inseguridad jurídica porque aumentan las posibilidades de violación accidental de derechos. Además, la creatividad es incremental; al crear siempre copiamos algo y la línea divisoria entre la copia burda y la inspiración es sutil. Para profundizar más en todo ello, revisemos algunas de las categorías de la propiedad intelectual.

3.1.1. Derechos de autor

Los derechos de autor (*copyright*) protegen la expresión de un contenido, no el contenido en sí mismo. Se desarrollaron para recompensar a los autores de libros o de arte. Las obras protegidas pueden expresar ideas, conocimientos o métodos libremente utilizables, pero se prohíbe reproducirlas sin permiso, total o parcialmente, con o sin modificaciones. Esta protección es muy sencilla, ya que entra automáticamente en vigor en el momento de publicación de la obra con ámbito casi universal. Modernamente, se ha extendido a los programas de ordenador y a recopilaciones de datos.

ANOTACIONES

La Ley de la Propiedad Intelectual (LPI), en España, y leyes similares en otros países, desarrolladas sobre la base del Convenio de Berna para la protección de trabajos literarios y artísticos de 1886, regulan los derechos de autor. Éstos se dividen en derechos morales y patrimoniales. Los primeros garantizan al autor el control sobre la divulgación de su obra, con nombre o seudónimo, el reconocimiento de autoría, el respeto a la integridad de la obra y el derecho de modificación y retirada. Los segundos dan derecho a explotar económicamente la obra y pueden ser cedidos total o parcialmente, de forma exclusiva o no, a un tercero. Los derechos morales son vitalicios o indefinidos, mientras que los patrimoniales tienen una duración bastante larga (70 años después de la muerte del autor, si es una persona física, en el caso de la ley española).

La cesión de derechos se especifica por un contrato denominado *licencia*. En el caso de programas propietarios, éstos generalmente se distribuyen por medio de licencias de uso *no exclusivo* que se entiende que se aceptan automáticamente al abrir o instalar el producto. No es necesario pues firmar el contrato, ya que, en el caso de no aceptarlo el receptor, rigen automáticamente los derechos por omisión de la ley, es decir, ninguno. Las licencias no pueden restringir algunos derechos, como el de hacer copias privadas de arte o música, lo que nos permite regalar una copia de una grabación a un amigo, pero este derecho no es aplicable a los programas. Según la LPI de 1996, de los programas siempre se puede hacer una copia de seguridad, se pueden estudiar para hacer programas compatibles y se pueden corregir y adaptar a nuestras necesidades (difícil, porque no solemos disponer de los códigos fuentes). Estos derechos no pueden ser restringidos por licencias, aunque las leyes están en proceso de revisión, en una tendencia aparentemente imparable a reducir los derechos de los usuarios. Las recopilaciones organizadas de obras o datos ajenos también están sometidas a derechos de autor, si bien los términos son distintos y la duración menor.

Lectura complementaria

[Ver bibliografía, apartado 61]

Las nuevas tecnologías de la información, y en especial *la Red*, han trastocado profundamente la protección de los derechos de autor, ya que las expresiones de contenidos son mucho más fáciles de copiar que los contenidos mismos. Y en el caso de los programas y algunas obras de arte (música, imágenes, películas, e incluso literatura) *funcionan* automáticamente en el ordenador, sin necesidad de un esfuerzo humano apreciable. En cambio, los diseños o inventos hay que cons-

ANOTACIONES

truirlos y, posiblemente, ponerlos en producción. Esta posibilidad de crear riqueza sin coste ha llevado a gran parte de la sociedad, en particular a los países pobres, a duplicar programas sin pagar licencia, no existiendo una conciencia social de que el actuar de este modo sea una *mala acción* (por contra, sí que la suele haber con respecto al robo de bienes físicos, por ejemplo). Por otro lado, los fabricantes de programas, solos o en coalición (p. ej.: la BSA o Business Software Alliance) presionan fuertemente para que las licencias se paguen y los gobiernos persigan lo que se ha dado en llamar *piratería*.

Nota

El término *piratería* se ha popularizado como sinónimo de violación de cualquier forma de propiedad intelectual, especialmente en el caso de la copia ilegal de programas, música y películas. El término parece exagerado, y en el diccionario de la Real Academia Española de la Lengua aparece como una acepción en sentido figurado, ya que el término original se refiere a robo con violencia en el mar. Por ello Richard Stallman recomienda evitarla.

Lectura complementaria

[Ver bibliografía, apartado 191]

3.1.2. Secreto comercial

Otro de los recursos de que disponen las empresas para rentabilizar sus inversiones es el secreto comercial, protegido por las leyes de propiedad industrial, siempre que las empresas tomen las medidas suficientes para ocultar la información que no quieren desvelar. En el caso de productos químicos o farmacéuticos que requieran aprobación gubernamental, el Estado se compromete a no desvelar los datos entregados que no sea obligatorio hacer públicos.

Una de las aplicaciones del secreto comercial más conocidas se encuentra en la industria del software propietario, que generalmente comercializa programas compilados sin dar acceso al código fuente, para impedir así el desarrollo de programas derivados.

A primera vista parece que la protección del secreto comercial es perversa, ya que puede privar indefinidamente a la sociedad de conocimientos útiles. En cierto modo así lo entienden algunas legisla-

ANOTACIONES

ciones, permitiendo la ingeniería inversa para desarrollar productos sustitutos, aunque la presión de las industrias ha conseguido que en muchos países ésta sea una actividad prohibida y en otros sólo esté permitida en aras de la compatibilidad.

Sea perverso o no el secreto comercial, en muchos casos es mejor que una patente, ya que permite una ventaja competitiva al que pone un producto en el mercado, mientras la competencia trata de imitarlo con ingeniería inversa. Cuanto más sofisticado sea el producto, más costará a la competencia reproducirlo, mientras que si es trivial, lo copiará rápidamente. La imitación con mejoras ha sido fundamental para el desarrollo de las que hoy son superpotencias (Estados Unidos y Japón) y es muy importante para la independencia económica de los países en desarrollo.

3.1.3. Patentes y modelos de utilidad

La alternativa al secreto comercial es la patente. A cambio de un monopolio de 17 a 25 años y un determinado coste económico, un *invento* es revelado públicamente, de forma que sea fácilmente reproducible. Con la patente se pretende promover la investigación privada, sin coste para el contribuyente y sin que el resultado se pierda. El poseedor de una patente puede decidir si permite a otros utilizarla y el precio que debe pagar por la licencia.

La doctrina oficial es que el sistema de patentes promueve la innovación, pero cada vez más se hacen oír voces que afirman que la dificulta, ya sea porque opinan que el sistema está mal implementado, ya porque creen que es perverso en sí mismo.

Lectura complementaria

[Ver bibliografía, apartados 75 y 173]

Lo que se considera un invento ha ido variando con el tiempo, con grandes presiones para ampliar la cobertura del sistema, incluyendo algoritmos, programas, modelos de negocio, sustancias naturales, genes y formas de vida, incluidas plantas y animales. TRIPS exige que el sistema de patentes no discrimine ningún ámbito del saber. Las presiones de la *Organización Mundial de la Propiedad Intelectual* (OMPI o WIPO) pretenden eliminar la necesidad de que el invento tenga aplicación industrial y también rebajar los estándares de inventiva exigibles en una patente. Estados Unidos está a la cabeza de los países con un mínimo de exigencias sobre lo que es patentable,

ANOTACIONES

siendo además el más beligerante para que otros países adopten sus estándares, sin acordarse que él mismo se negó a aceptar las patentes extranjeras cuando era un país subdesarrollado.

Una vez obtenida una patente, los derechos del poseedor son independientes de la calidad del invento y del esfuerzo invertido en obtenerlo. Dado el coste de mantenimiento de una patente y los costes de litigación, solamente las grandes empresas pueden mantener y mantienen una amplia cartera de patentes que la sitúan en una posición que les permite ahogar cualquier competencia. Dada la facilidad para colocar patentes sobre soluciones triviales o de muy amplia aplicabilidad, pueden monopolizar para sí un espacio muy amplio de actividad económica.

Con patentes, muchas actividades, especialmente la programación, se hacen extremadamente arriesgadas, ya que es muy fácil que en el desarrollo de un programa complicado se viole accidentalmente alguna patente. Cuando dos o más empresas están investigando para resolver un problema, es muy probable que lleguen a una solución similar casi al mismo tiempo, pero sólo una (generalmente la de más recursos) logrará patentar su invento, perdiendo las otras toda posibilidad de rentabilizar su inversión. Todo desarrollo técnico complejo puede convertirse en una pesadilla si para cada una de las soluciones de sus partes es necesario investigar si la solución encontrada está patentada (o en trámite), para intentar obtener la licencia o para buscar una solución alternativa. Este problema deviene especialmente grave en el software libre, donde las violaciones de patentes de algoritmos son evidentes por simple inspección del código. Aunque en Europa aún es ilegal patentar un algoritmo, puede serlo en muy breve plazo, quizá cuando el lector lea estas líneas.

3.1.4. Marcas y logotipos

Son nombres y símbolos que representan un acervo de calidad (o una gran inversión en propaganda). No tienen gran importancia dentro del software libre, posiblemente porque supone un coste registrarlas. Así, solamente algunos nombres importantes, como Open Source (por Open Source Foundation), Debian, (por Software in the Public Interest), GNOME (GNOME Foundation), GNU (Free Software Foundation) están registrados, y sólo en algunos países. Sin embargo, el no registro de

ANOTACIONES

nombres ha provocado problemas. Por ejemplo, en Estados Unidos (1996) y en Corea (1997) ha habido personas que han registrado el nombre Linux y han demandado dinero por su uso. La resolución de estas disputas supone costes legales y la necesidad de demostrar el uso del nombre anterior a la fecha del registro.

3.2. Licencias en el software libre

Estrictamente hablando, lo que diferencia al software libre del resto del software es un aspecto legal: la *licencia*. Se trata, en palabras de uso común, de un *contrato* entre el autor (o propietario de los derechos) y los usuarios, que estipula lo que los éstos pueden hacer con su obra: uso, redistribución, modificación, etc., y en qué condiciones.

Aunque en esencia software libre y software propietario se diferencien en la licencia con la que los autores publican sus programas, es importante hacer hincapié en que las diferencias entre las diferentes licencias, aunque puedan parecer nimias, suelen suponer condiciones de uso y redistribución totalmente diferentes y, como se ha podido demostrar a lo largo de los últimos años, han desembocado no sólo en métodos de desarrollo totalmente diferentes, sino incluso en una forma alternativa de entender la informática.

La legislación sobre derechos de autor, plasmada en las leyes sobre propiedad intelectual, asegura que *por defecto* no se puede hacer casi nada con una obra (en nuestro caso, un programa) que se recibe o se compra si su autor (o el que posea los derechos de la misma) no nos lo permite explícitamente. Las licencias de software libre dan ciertos permisos explícitos: Cuando recibes un programa libre puedes redistribuirlo o no, pero si lo redistribuyes, sólo puedes hacerlo porque la licencia te lo permite. Pero para ello es preciso cumplir con la licencia... En definitiva, la licencia contiene las normas de uso a las que han de atenerse usuarios, distribuidores, integradores y otras partes implicadas en el mundo de la informática.

Las condiciones y/o restricciones que imponen las licencias sólo pueden ser precisadas por los propios autores, que según la normativa de propiedad intelectual son los propietarios de la obra. En cualquier caso, la propiedad de la obra será de los autores, ya que la licencia no supone

ANOTACIONES

transferencia de propiedad, sino solamente derecho de uso y, en algunos casos, de distribución.

Para comprender plenamente todos los entresijos legales que se van a presentar en este capítulo (y que, sin duda, son muy importantes para entender la naturaleza del software libre) también es necesario saber que cada nueva versión de un programa es considerada como una nueva obra. El autor tiene, otra vez, plena potestad para hacer con su obra lo que le apetezca, incluso distribuirla con términos y condiciones totalmente diferentes (o sea, una licencia diferente a la anterior). Así, si el lector es autor único de un programa podrá publicar una versión bajo una licencia de software libre y, si le apeteciere, otra posterior bajo una licencia propietaria. En caso de existir más autores, y que la nueva versión contenga código cuya autoría les corresponda y que se vaya a publicar bajo otras condiciones, todos ellos han de dar el visto bueno al cambio de licencia.

Un tema todavía por discernir, y por tanto abierto, es conocer la licencia que cubre a las contribuciones *externas*. Generalmente, se supone que una persona que contribuya al proyecto acepta *de facto* que su contribución se ajuste a las condiciones especificadas por la licencia del proyecto, aunque suponemos que ello puede tener poco respaldo jurídico. La iniciativa de la Free Software Foundation de pedir mediante carta (física) la cesión de todos los derechos de copyright a cualquiera que contribuya con más de diez líneas de código a un subproyecto de GNU es una buena muestra de que esto no tiene por qué ser así.

Partiendo de todo lo dicho, vamos a centrarnos ya el resto de este capítulo en el análisis de diversas licencias. Para poner en contexto este estudio, es preciso recordar que de ahora en adelante cuando decimos que una licencia es de software libre, lo decimos en el sentido de que cumple las definiciones de software libre que se presentaron en el apartado 1.1.1.

3.2.1. Licencias tipo BSD

La licencia BSD (Berkeley Software Distribution) tiene su origen en la publicación de versiones de UNIX realizadas por la universidad californiana de Berkeley, en EE.UU. La única obligación que exige es la de dar crédito a los autores, mientras que permite tanto la redistribución bina-

ANOTACIONES

ria, como la de loscódigos fuentes, aunque no obliga a ninguna de las dos en ningún caso. Asimismo, da permiso para realizar modificaciones y ser integrada con otros programas casi sin restricciones.

La licencia BSD es ciertamente muy popular, como se puede ver a partir del hecho de que existen varias licencias de condiciones similares (XWindow, Tcl/Tk, Apache), que se han venido a llamar licencias tipo BSD. Estas licencias reciben el nombre de minimalistas, ya que las condiciones que imponen son pocas, básicamente asignar la autoría a los autores originales. Su concepción se debe al hecho de que el software publicado bajo esta licencia era software generado en universidades con proyectos de investigación financiados por el gobierno de los Estados Unidos; las universidades prescindían de la comercialización del software creado, ya que asumían que ya había sido pagado previamente por el gobierno, y por tanto con los impuestos de todos los contribuyentes, por lo que cualquier empresa o particular podía utilizar el software casi sin restricciones, incluso redistribuyendo modificaciones al mismo de manera binaria sin tener que entregar las fuentes.

Este último punto hace que a partir de un programa distribuido bajo una licencia de tipo BSD pueda crearse otro programa (en realidad otra versión del programa) propietario, o sea, que se distribuyera con una licencia más restrictiva. Los críticos de las licencias BSD ven en esta característica un peligro, ya que no se garantiza la libertad de versiones futuras de los programas. Los partidarios de la misma, por contra, ven en ella la máxima expresión de la libertad y argumentan que, a fin de cuentas, se puede hacer (casi) lo que se quiera con el software.

Nota

Una de las consecuencias prácticas de las licencias tipo BSD ha sido la difusión de estándares, ya que los desarrolladores no encuentran ningún obstáculo para realizar programas compatibles con una implementación de referencia bajo este tipo de licencia. De hecho, ésta es una de las razones de la extraordinaria y rápida difusión de los protocolos de Internet y de la interfaz de programación basada en *sockets*, ya que la mayoría de los desarrolladores comerciales derivó su realización de la de la Universidad de Berkeley.

ANOTACIONES

Entre las licencias del tipo BSD podemos encontrar la de XWindow, Tcl/Tk y Apache. La mayoría de ellas son una copia calcada de la original de Berkeley, modificando todo lo referente a la autoría. Otras, como la Apache, incluyen alguna cláusula adicional, como la imposibilidad de llamar las versiones redistribuidas de igual manera. Todas suelen incluir, como ella, la prohibición de usar el nombre del propietario de los derechos para promocionar productos derivados.

Asimismo, todas las licencias, sean de tipo BSD o no, incluyen una *limitación de garantía* que es en realidad una *negación de garantía*, necesaria para evitar demandas legales por garantías implícitas. Aunque se ha criticado mucho esta negación de garantía en el software libre, es práctica habitual en el software propietario, que generalmente sólo garantiza que el soporte es correcto y el programa en cuestión se ejecuta.



Esquema resumen de la licencia BSD: Copyright © *el propietario*. Todos los derechos reservados.

Se permite la redistribución en fuente y en binario con o sin modificación, siempre que se cumplan las condiciones siguientes:

1. Las redistribuciones en fuente deben retener la nota de *copyright* y listar estas condiciones y la limitación de garantía,
2. Las redistribuciones en binario deben reproducir la nota de *copyright* y listar estas condiciones y la limitación de garantía en la documentación.
3. Ni el nombre del *propietario* ni de los que han contribuido pueden usarse sin permiso para promocionar productos derivados de este programa.

ESTE PROGRAMA SE PROPORCIONA *TAL CUAL*, SIN GARANTÍAS EXPRESAS NI IMPLÍCITAS, TALES COMO SU APLICABILIDAD COMERCIAL O SU ADECUACIÓN

ANOTACIONES

PARA UN PROPÓSITO DETERMINADO. EN NINGÚN CASO *EL PROPIETARIO* SERÁ RESPONSABLE DE NINGÚN DAÑO CAUSADO POR SU USO (INCLUYENDO PÉRDIDA DE DATOS, DE BENEFICIOS O INTERRUPCIÓN DE NEGOCIO).

3.2.2. La Licencia Pública General de GNU (GNU GPL)

La *Licencia Pública General del proyecto GNU* (más conocida por su acrónimo en inglés GPL) es con diferencia la licencia más popular y conocida de todas las licencias del mundo del software libre. Su autoría corresponde a la Free Software Foundation (promotora del proyecto GNU) y en un principio fue creada para ser la licencia de todo el software generado por la FSF. Sin embargo, su utilización ha ido más allá hasta convertirse en la licencia más utilizada (más del 70% de los proyectos anunciados en FreshMeat están licenciados bajo la GPL), incluso por proyectos bandera del mundo del software libre, como es el caso del núcleo Linux.

La licencia GPL es interesante desde el punto de vista legal porque hace un uso tan curioso de la legislación de *copyright* que haciendo estricto uso del término llega a una solución totalmente contraria a la original, hecho por el que también se ha venido a llamar una licencia *copyleft*. Alguien, con una pizca de humor, llegó incluso a lanzar el eslogan *copyleft, all rights reversed*.

En líneas básicas, la licencia GPL permite la redistribución binaria y la de las fuentes, aunque, en el caso de que redistribuya de manera binaria, obliga a que también se pueda acceder a las fuentes. Asimismo, está permitido realizar modificaciones sin restricciones, aunque sólo se pueda integrar código licenciado bajo GPL con otro código que se encuentre bajo una licencia idéntica o compatible, lo que ha venido a llamarse el efecto *viral* de la GPL, ya que el código publicado una vez con esas condiciones nunca puede cambiar de condiciones.

ANOTACIONES

Nota

Una licencia es incompatible con la GPL cuando restringe alguno de los derechos que la GPL garantiza, ya sea explícitamente contradiciendo alguna cláusula, ya implícitamente, imponiendo alguna nueva. Por ejemplo, la licencia BSD actual es compatible, pero la original (o la de Apache), que exige en los materiales de propaganda que se mencione explícitamente que el trabajo combinado contiene código de todos y cada uno de los titulares de derechos, la hace incompatible.

La licencia GPL está pensada para asegurar la libertad del código en todo momento, ya que un programa publicado y licenciado bajo sus condiciones nunca podrá ser hecho propietario. Es más, ni ese programa ni modificaciones al mismo pueden ser publicados con una licencia diferente a la propia GPL. Los partidarios de las licencias tipo BSD ven en esta cláusula un recorte de la libertad, mientras que sus seguidores ven en ello una forma de asegurarse que ese software siempre va a ser libre. Por otro lado, se puede considerar que la licencia GPL maximiza las libertades de los usuarios, mientras que las de tipo BSD lo hacen para los desarrolladores. Nótese, sin embargo, que en el segundo caso estamos hablando de los desarrolladores en general y no de los autores, ya que muchos autores consideran que la licencia GPL es más beneficiosa para sus intereses, ya que obliga a sus *competidores* a publicar sus modificaciones (mejoras, correcciones, etc.) en caso de redistribuir el software, mientras que con una licencia tipo BSD éste no tiene por qué ser el caso. De cualquier modo, podemos ver que la elección de licencia no es una tarea fácil y que hay que tener multitud de factores en cuenta.

En cuanto a la naturaleza contraria al *copyright*, esto se debe a que la filosofía que hay detrás de esta licencia (y detrás de la Free Software Foundation) es que el software no debe tener propietarios. Aunque es cierto que el software licenciado con la GPL tiene un autor, que es el que a fin de cuentas permite la aplicación de la legislación de *copyright* sobre su obra, las condiciones bajo las que publica su obra confieren a la misma tal carácter que podemos considerar que la propiedad del software corresponde a quien lo tiene y no a quien lo ha creado.

ANOTACIONES

Por supuesto, también incluye *negaciones de garantía* para proteger a los autores. Asimismo, y para proteger la buena fama de los autores originales, toda modificación de un fichero fuente debe incluir una nota con la fecha y autor de cada modificación.

La GPL contempla también a las patentes de software, exigiendo que si el código lleva algoritmos patentados (como dijimos, algo legal y usual en Estados Unidos y práctica irregular en Europa), o se concede licencia de uso de la patente libre de tasas, o no se puede distribuir bajo la GPL.

La licencia GPL se encuentra en la actualidad, y desde hace más de diez años, en su segunda versión y es posible que en próximas fechas se publique una tercera. Generalmente, esto no supone mayor problema, ya que la mayoría de los autores suelen publicar los programas bajo las condiciones de la segunda versión de la GPL o cualquier posterior publicada por la Free Software Foundation. Como anécdota, cabe decir que Linus Torvalds, creador de Linux, publica su software sólo bajo las condiciones de la segunda versión de la GPL, buscando desmarcarse de las posturas –a veces demasiado radicales, según él– de la Free Software Foundation.

3.2.3. La Licencia Pública General Menor de GNU (GNU LGPL)

La *Licencia Pública General Menor del proyecto GNU* (comúnmente conocida por sus iniciales en inglés LGPL) es la otra licencia de la Free Software Foundation. Pensada en sus inicios para su uso en bibliotecas (la L en sus comienzos venía de *library*, 'biblioteca'), fue modificada recientemente para ser considerada la hermana menor (*lesser,*'menor') de la GPL.

Lectura complementaria

[Ver bibliografía, apartado 97]

La LGPL permite el uso de programas libres con software propietario. El programa en sí se redistribuye como si estuviera bajo la licencia GPL, pero se permite la integración con cualquier otro software sin prácticamente limitaciones.

Como se puede ver, en un principio esta licencia estaba orientada a las bibliotecas, de manera que se pudiera potenciar su uso y desarrollo sin tener los problemas de integración que implica la GPL. Sin embar-

ANOTACIONES

go, cuando se vio que el efecto buscado de popularizar las bibliotecas libres no se veía compensado por la generación de programas libres, la Free Software Foundation decidió el cambio de *library* a *lesser* y desaconsejó su uso, salvo para condiciones muy puntuales y especiales. Hoy en día, existen muchos programas que no son bibliotecas licenciados bajo las condiciones de la LGPL. Por ejemplo, el navegador Mozilla está licenciado, entre otras, también bajo la LGPL.

3.2.4. Otras licencias de programas

La variedad de licencias libres es grande, aunque por razones prácticas la mayoría de los proyectos se adaptan a las descritas anteriormente: en efecto muchos proyectos no quieren o pueden dedicar recursos a diseñar una licencia propia y la mayoría de los usuarios prefieren referirse a unas siglas, que leer y analizar licencias completas. Que una licencia sea libre o no, depende del concepto de libertad que tenga el que la clasifica, aunque existe consenso para la mayoría de ellas.



Sugerencia

Puede ver recopiladas y comentadas tanto licencias consideradas libres, como licencias consideradas no libres o libres pero incompatibles con la GPL desde el punto de vista de la FSF en [fsf:licences]. El punto de vista filosóficamente diferente de la *Open Source Initiative* se refleja en su listado[osi:licenses]. Pueden verse discrepancias en algunas licencias, como la Apple Public Source License Ver. 1.2, considerada no libre por la FSF por la obligación de publicar todos los cambios (aunque sean privados), notificar a Apple de las redistribuciones, o por la posibilidad de revocación unilateral. No obstante, la presión de esta clasificación ha hecho que Apple publique la versión 2.0 en agosto del 2003, ya considerada libre por la FSF.

A modo de ejemplo, se incluyen a continuación dos pequeños listados con algunas licencias de software libre no tratadas hasta el momento.

ANOTACIONES

En el primero pueden encontrarse licencias que no imponen condiciones especiales en la *segunda* redistribución (esto es, que sólo especifican que el software se puede redistribuir o modificar, pero no imponen condiciones especiales si se hace, lo que permite, por ejemplo, que alguien que recibe el programa pueda después redistribuirlo como software propietario). En el segundo se incluyen las que, al estilo de la GNU GPL, imponen condiciones en caso de que se quiera redistribuir el software, condiciones que van en la línea de forzar a que se sigan cumpliendo las condiciones de la licencia después de la *primera redistribución*. Mientras que el primer grupo hace énfasis en la libertad de quien recibe un programa, ya que le permite hacer casi lo que quiera con él (en términos de condiciones de futuras redistribuciones), el segundo lo hace en la libertad de cualquiera que potencialmente pueda recibir algún día un trabajo derivado del programa (ya que obliga a que las sucesivas modificaciones y redistribuciones respeten los términos de la licencia original). La diferencia entre estos dos tipos de licencias ha sido (y es) tema de debate en la comunidad del software libre. En el resto de este apartado llamaremos "licencias permisivas" o "licencias tipo BSD" a las del primer tipo, y "licencias robustas" o "licencias *copyleft*"a las del segundo. En cualquier caso, es conveniente recordar que todas ellas son licencias libres.

Nota

El término *copyleft* aplicado a una licencia, usado sobre todo por la Free Software Foundation para definir sus licencias, tiene implicaciones similares a las del término "licencia robusta", tal y como lo usamos en este texto.

Algunas licencias permisivas:

- Licencia de X Window versión 11 (X11)

Es la licencia usada para la distribución del sistema X Window, el sistema de ventanas más ampliamente usado en el mundo Unix, y también en entornos GNU/Linux. Es una licencia muy similar a la licencia BSD, que permite redistribución, uso y modificación prácticamente sin restricciones. A veces, esta licencia es llamada "licencia MIT" (con peligrosa poca precisión, porque el MIT ha usado otros tipos de licencias). Bajo esta licencia se distribuyen también trabajos derivados de X Windows, como XFree86.

Lectura complementaria

[Ver bibliografía, apartados 57 y 60]

ANOTACIONES

- Zope Public License 2.0

Esta licencia (habitualmente llamada “ZPL”) es usada para la distribución de Zope (un servidor de aplicaciones) y otros productos relacionados. Es una licencia similar a la BSD, con el interesante detalle de prohibir expresamente el uso de marcas registradas por Zope Corporation.

- Licencia de Apache

Licencia bajo la que se distribuyen la mayor parte de los programas producidos por el proyecto Apache. Es similar a la licencia BSD.

Hay algunos programas libres que no se distribuyen con una licencia específica, sino que su autor los declara explícitamente *public domain* (en el dominio público, o del común). La principal consecuencia de esta declaración es que el autor renuncia a todos sus derechos sobre el programa, y por lo tanto puede modificarse, redistribuirse, usarse, etc. de cualquier manera. A efectos prácticos, esta situación es muy similar a que el programa esté bajo una licencia tipo BSD.

Lectura complementaria

[Ver bibliografía, apartados 49, 50, 21, 20, 62, 34 y 39]

Algunas licencias robustas:

- Licencia de Sleepycat

Es la licencia bajo la que la empresa Sleepycat distribuye sus programas (entre ellos el conocido Berkeley DB). Obliga a ciertas condiciones siempre que se redistribuye el programa o trabajos derivados del programa. En particular, obliga a ofrecer el código fuente (incluidas las modificaciones, si se trata de un trabajo derivado), y a que la redistribución imponga al receptor las mismas condiciones. Aunque mucho más corta que la GNU GPL, es muy similar a ésta en sus efectos principales.

- eCos License 2.0

Es la licencia bajo la que se distribuye eCos, un sistema operativo de tiempo real. Es una modificación de la GNU GPL, que no considera

ANOTACIONES

que el código que se enlace con programas protegidos por ella queden sujetos a las cláusulas de la GNU GPL si se redistribuyen. Desde este punto de vista, sus efectos son similares a los de la GNU LGPL.

- Affero General Public License

Interesante modificación de la GNU GPL, que considera el caso de los programas que ofrecen servicios vía web, o en general, vía redes de ordenadores. Este tipo de programas plantean un problema desde el punto de vista de las licencias *robustas*. Como el uso del programa no implica haberlo recibido mediante una redistribución, aunque el programa esté licenciado, por ejemplo, bajo la GNU GPL, alguien puede modificarlo y ofrecer un servicio en la red usándolo, sin redistribuirlo de ninguna forma, y por tanto sin estar obligado, por ejemplo, a distribuir el código fuente. La Affero GPL tiene una cláusula que obliga a que, si el programa tiene un medio para proporcionar su código fuente vía web a quien lo use, no se pueda desactivar esa característica. Esto significa que, si el autor original incluye esa capacidad en el código fuente, cualquier usuario puede obtenerlo, y además esa *redistribución* está sometida a las condiciones de la licencia. La Free Software Foundation está considerando incluir provisiones similares en la versión 3 de su GNU GPL.

- IBM Public License Version 1.0

Es una licencia que permite la redistribución binaria de trabajos derivados sólo si (entre otras condiciones) se prevé algún mecanismo para que quien reciba el programa pueda recibir su código fuente. La redistribución del código fuente se ha de hacer bajo la misma licencia. Además, esta licencia es interesante por obligar al que redistribuye el programa con modificaciones a licenciar automática y gratuitamente las patentes que puedan afectar a esas modificaciones, y que sean propiedad del redistribuidor, a quien reciba el programa.

- Mozilla Public License 1.1

Ejemplo de licencia libre con origen en una empresa. Es una evolución de la primera licencia libre que tuvo el Netscape Navigator, y en su momento fue muy importante por ser la primera vez que una em-

ANOTACIONES

presa muy conocida decidió distribuir un programa bajo su propia licencia libre.

3.2.5. Distribución bajo varias licencias

Hasta ahora hemos ido suponiendo que cada programa tenía una única licencia en la que se especificaban las condiciones de uso y redistribución, entre otras. Sin embargo, un autor puede distribuir obras con distintas licencias. Para entenderlo, debemos tener en cuenta que cada publicación es una nueva obra y que se puede dar el caso de que se distribuyan versiones que sólo difieren en la licencia. Como veremos, en la mayoría de los casos esto se traduce en que, dependiendo de lo que el usuario quiera hacer con el software, se encontrará con que tiene que obedecer una licencia u otra.

El ejemplo más conocido de doble licencia es el de la biblioteca Qt, sobre la que se fundamenta el entorno de escritorio KDE. Trolltech, una empresa afincada en Noruega, distribuía Qt con una licencia propietaria, aunque eximía del pago a los programas que hicieran uso de la misma sin ánimo de lucro. Por esta causa y por su potencia técnica fue elegida a mediados de la década de los noventa por el proyecto KDE, lo que supuso una ardua polémica con la Free Software Foundation, ya que KDE dejaba de ser entonces software libre en su conjunto, al depender de una biblioteca propietaria. Tras un largo debate –en el que, entre otras cosas, apareció GNOME como competidor libre de KDE en el escritorio– Trolltech decidió utilizar el sistema de doble licencia para su producto estrella: los programas bajo GPL podían hacer uso de una versión de Qt GPL, mientras que los que no lo eran, debían comprarles una licencia. Esta solución satisfizo a todas las partes, por lo que hoy en día KDE es considerado software libre.

Otros ejemplos de licencias duales los podemos encontrar en StarOffice y OpenOffice.org, o en el ya difunto Netscape Communicator y Mozilla. En ambos casos el primer producto es propietario, mientras que el segundo es una versión libre (generalmente bajo las condiciones de varias licencias libres). Aunque en un principio los proyectos libres eran versiones limitadas de sus hermanos propie-

ANOTACIONES

tarios, con el tiempo han ido tomando su propio camino, por lo que a día de hoy tienen un grado de independencia bastante grande.

3.3. Licencias de otros recursos libres

Las licencias de software libre han sido fuente de inspiración para otros recursos intelectuales, de tal modo que muchos de ellos la han adoptado directamente, especialmente en el caso de la documentación, en otros casos las han adaptado ligeramente, como es el caso de la pionera Open Audio License. La mayoría de ellas son de tipo *copyleft* si permiten trabajos derivados.

Lectura complementaria

[Ver bibliografía, apartado 94]

Incluso se han utilizado licencias de programas (GPL y LGPL) para hardware, aunque esta materia es compleja y difícil de conciliar con la legalidad vigente. En efecto, los diseños y diagramas pueden ser utilizados, sin copiarlos físicamente, para extraer ideas que se utilicen para nuevos diseños cerrados. Por ejemplo, la *OpenIPCore Hardware General Public License* contempla la prohibición de esta apropiación, pero su legalidad es dudosa. La única posibilidad de proteger esas ideas es por medio de algún tipo de patente libre, algo aún no desarrollado y fuera del alcance de los que no tienen intención o posibilidad de hacer negocio con ellas.

Lectura complementaria

[Ver bibliografía, apartados 135 y 188]

3.3.1. Licencia de documentación libre de GNU

Una de las licencias tipo *copyleft* más conocidas para la documentación técnica, ya sea de programas o de cualquier otra cosa, es la de la Free Software Foundation. Después de darse cuenta de que un documento no es lo mismo que un programa, Richard Stallman promovió una licencia para los documentos que acompañan a los programas y para otros documentos de carácter técnico o didáctico.

Para facilitar el desarrollo de versiones derivadas, hay que poner a disposición de quien lo requiera una copia *transparente* del documento, en el sentido del apartado 1.4.1 y definida en el apéndice B, apartado B.2, además de las copias *opacas*, en una analogía ente los códigos fuentes y los objetos de los programas.

ANOTACIONES

Una de las preocupaciones de la licencia es reconocer la autoría e impedir que se tergiversen ideas u opiniones expresadas por el autor. Para ello, se exige que las obras derivadas exhiban en la portada un título distinto a los de las versiones anteriores (salvo permiso expreso) y se nombre expresamente de dónde se puede conseguir el original. También deben listarse como autores los más importantes de los originales –además de los autores de las modificaciones– y conservarse todas las notas sobre derechos de autor. Deben conservarse, asimismo, agradecimientos, dedicatorias, así como respetarse el apartado de historia, si lo tiene, añadiendo las modificaciones nuevas. Incluso, y esto es lo más criticado de la licencia, pueden nombrarse secciones invariantes y textos de cubiertas, que nadie puede modificar ni eliminar, si bien la licencia sólo permite considerar invariante textos *no técnicos*, llamados *secundarios* por la misma.

Esta licencia ha generado gran polémica en el mundo del software libre, hasta el punto de que en la distribución Debian se está discutiendo (en el momento de publicar este libro) si eliminar o pasar a la sección extraoficial *no libre* todo documento con la misma. Incluso aunque no existan secciones invariantes, al regirse los trabajos derivados por la misma licencia, las pueden añadir. Se argumenta, por ejemplo, que algunas secciones invariantes pueden ser incorrectas u estar obsoletas, pero deben mantenerse. En todo caso la licencia es incompatible con las directrices de software libre de Debian, pero la cuestión quizá esté en si la documentación debe cumplirlas (por ejemplo, los textos de las licencias tampoco se pueden modificar).



Sugerencia

Este texto está amparado en esta licencia, por lo que se incluye su versión original en el apéndice A, y una traducción en el apéndice B. Estudiar cualquiera de las versiones y formarse una opinión.

3.3.2. Licencias de Creative Commons

En el 2001 se fundó *Creative Commons*, dirigido por expertos en propiedad intelectual, derecho en la sociedad de la información, e infor-

Lectura complementaria

[Ver bibliografía, apartados 18 y 154]

ANOTACIONES

mática, con el propósito de fomentar la existencia, conservación y accesibilidad de recursos intelectuales cedidos a la comunidad de diversas maneras. Uno de sus proyectos más conocidos fue el desarrollo, a finales del 2002 de una serie de licencias concebidas, no para software, sino para trabajos literarios, artísticos, didácticos, etc. Su característica más sobresaliente, además de estar avaladas por profesionales del derecho, es que permiten al autor seleccionar qué tipo de libertades cede, además de la de copia, según cuatro dimensiones: dar crédito al autor original, permitir trabajos derivados, permitir redistribución comercial y permitir cambiar la licencia. Así, por ejemplo, la licencia de los cursos del MIT (MIT Open Courseware License Version 1.0) está basada en la de Creative Commons que obliga a dar crédito, impide el uso comercial y obliga a conservar la licencia en trabajos derivados.



A pesar de haber 4 dimensiones, no hay 16 licencias, sino 11 solamente. En la tabla siguiente se nombran y se muestran los símbolos utilizados para designarlas gráficamente:

Attribution				
Attribution-NoDerivs				
Attribution-NoDerivs-NonCommercial				
Attribution-NonCommercial				
Attribution-NonCommercial-ShareAlike				
Attribution-ShareAlike				
NoDerivs				

¿Por qué no existen las 16?

3.4. Resumen

En este capítulo hemos podido ver la importancia que tienen las licencias dentro del mundo del software libre y de los demás recursos

ANOTACIONES

libres. Asimismo, hemos presentado la gran variedad de licencias existentes, su motivación, sus repercusiones y sus ventajas e inconvenientes. En definitiva, podemos decir que la GPL trata de maximizar las libertades que tiene el usuario del software –lo reciba directamente de su autor o no– mientras que las licencias tipo BSD lo que hacen es maximizar las libertades del modificador o redistribuidor.

A la vista de lo que se ha comentado en este capítulo, se deduce que es muy importante decidir pronto qué licencia va a tener un proyecto y conocer detalladamente sus ventajas e inconvenientes, ya que una modificación posterior suele ser muy difícil, sobre todo si el número de contribuciones externas es muy grande.

Para finalizar, queremos hacer hincapié en el hecho de que el software libre y el software propietario se diferencien de manera estricta única y exclusivamente en la licencia con la que se publican los programas. En próximos capítulos veremos, sin embargo, que esta puntualización meramente legal puede tener consecuencias –o puede que no– en la manera como se desarrolla el software, dando lugar a un nuevo modelo de desarrollo que se diferencie en mayor o menor medida, según el caso, de los métodos de desarrollos *tradicionales* utilizados en la industria del software.

4. El desarrollador y sus motivaciones

4.1. Introducción

El desarrollo parcialmente anónimo y distribuido del software libre ha permitido que durante muchos años los recursos humanos con los que cuenta el software libre sean desconocidos. Consecuencia de este desconocimiento ha sido la mistificación, al menos parcial, del mundo del software libre y de la vida de los que están detrás de él, amparándose en tópicos más o menos extendidos sobre la cultura *hacker* y los ordenadores. Desde hace unos pocos años, se ha venido realizando un gran esfuerzo por parte de la comunidad científica para conocer mejor a las personas que participan en proyectos de software libre, su procedencia, sus motivaciones, su preparación y otros aspectos que pudieran parecer interesantes. Desde el punto de vista puramente pragmático, conocer quién se implica y por qué en este tipo de proyectos puede ser de gran utilidad para la generación de software libre. Algunos científicos, principalmente economistas, psicólogos y sociólogos, han querido ir más allá y han visto en esta comunidad el germen de futuras comunidades virtuales con reglas y jerarquías propias, en muchos casos totalmente diferentes a las que conocemos en la sociedad *tradicional*. Entre las incógnitas más importantes está la de conocer los motivos que llevan a los desarrolladores a ser partícipes en una comunidad de estas características, habida cuenta de que los beneficios económicos, al menos los directos, son prácticamente inexistentes, mientras los indirectos son difícilmente cuantificables.

4.2. ¿Quiénes son los desarrolladores?

Este apartado pretende ofrecer una visión global de las personas que dedican su tiempo y su esfuerzo a participar en proyectos de software libre. Los datos que se van a mostrar provienen en su mayoría de estudios científicos realizados en los últimos años, siendo los más significativos –aunque, por supuesto, no los únicos.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartados 104 y 176]

Los desarrolladores de software libre son generalmente personas jóvenes. La media de edad está situada en torno a los 27 años. La varianza de la edad es muy grande, ya que el grupo predominante se encuentra en una horquilla que va desde los 21 a los 24 años, siendo la mediana –el valor que aparece con mayor frecuencia– los 23 años. Es interesante observar cómo la edad de incorporación al movimiento de software libre tiene sus máximos entre los 18 y 25 años –siendo especialmente pronunciada entre los 21 y 23 años–, lo que equivaldría a la edad universitaria. Esta evidencia contrasta con la afirmación de que el software libre es cosa principalmente de adolescentes, aunque su presencia es evidente (alrededor de un 20% de los desarrolladores tiene menos de 20 años). En definitiva, podemos ver cómo los desarrolladores suelen ser mayoritariamente veinteañeros (un 60%), mientras que los menores de 20 y los mayores de 30 se reparten a partes iguales el 40% restante.

De la edad de incorporación se puede deducir que existe una gran influencia universitaria en el software libre. Esto no es de extrañar, ya que, como se ha podido ver en el capítulo de historia, el software libre –antes incluso de llevar esta denominación– ha estado íntimamente ligado a las instituciones educativas superiores. Aún hoy, el verdadero motor del uso y expansión del software libre siguen siendo las universidades y los grupos de usuarios estudiantiles. No es, por tanto, de extrañar que más de un 70% de los desarrolladores cuenten con una preparación universitaria. El dato tiene mayor importancia si tenemos en cuenta que del 30% restante muchos no son universitarios porque todavía están en su fase escolar. Aun así, también tienen cabida –y no por ello son menos apreciados– desarrolladores que no han accedido nunca a estudios superiores, pero que son amantes de la informática.

El desarrollador de software libre es generalmente varón. Las cifras que se manejan en las diferentes encuestas sobre la presencia de mujeres en la comunidad varían entre un 1% y un 3%, compitiendo con el propio margen de error de las encuestas. Por otro lado, una mayoría (60%) afirma tener pareja, mientras que el número de desarrolladores con hijos sólo es de un 16%. Dados los márgenes de edades en los que están comprendidos los desarrolladores de software libre, estos datos concuerdan bastante bien con una muestra aleatoria, por lo que se pueden considerar *normales*. El mito del de-

ANOTACIONES

sarrollador solitario, cuya afición por la informática es lo único en su vida, se muestra, como puede verse, como una excepción a la regla.

4.3. ¿Qué hacen los desarrolladores?

Profesionalmente, los desarrolladores de software libre se definen como ingenieros software (33%), estudiantes (21%), programadores (11%), consultores (10%), profesores de universidad (7%), etc. En el lado contrario, podemos ver cómo no suelen integrar ni los departamentos comerciales, ni de marketing (alrededor de un 1%). Es interesante observar cómo muchos de ellos se definen a sí mismos como ingenieros software antes que programadores –casi tres veces más–, teniendo en cuenta, como se verá en el capítulo dedicado a la ingeniería del software, que la aplicación de las técnicas clásicas de ingeniería de software (e incluso algunas modernas) no suele estar muy arraigada en el mundo del software libre.

El vínculo universitario que ya ha sido mostrado con anterioridad vuelve a aparecer en este apartado. Alrededor de uno de cada tres desarrolladores es estudiante o profesor de universidad, lo que viene a demostrar que existe una gran colaboración entre gente proveniente principalmente de la industria del software (los dos tercios restantes) y el ámbito académico.

Por otro lado, también se ha podido constatar una gran interdisciplinariedad: uno de cada cinco desarrolladores proviene de campos diferentes al de las tecnologías de la información. Esto, unido al hecho de que existe también un número similar de desarrolladores no universitarios, refleja la existencia de una gran riqueza en cuanto a intereses, procedencias y, en definitiva, a la composición de los equipos de desarrollo. Es muy difícil encontrar una industria moderna donde el grado de heterogeneidad sea tan grande como el que se puede ver en el software libre, si es que existe.

Además de aproximadamente el 20% de estudiantes, los desarrolladores suelen ser en su gran mayoría asalariados (64%), mientras que el porcentaje de autónomos es del 14%. Finalmente, sólo un 3% dice

ANOTACIONES

encontrarse en paro, siendo este dato significativo, ya que la encuesta fue hecha tras el comienzo de la crisis de las puntocom.

Nota

El hecho de que la financiación del software libre, al contrario que pasa con el propietario, no pueda hacerse mediante la venta de licencias ha propiciado desde siempre acalorados debates en torno a cómo deben ganarse la vida los programadores. En las encuestas que se están comentando en este capítulo, más de un 50% de los desarrolladores decían haberse beneficiado económicamente de manera directa o indirecta de su implicación en el software libre. Sin embargo, hay muchos que no lo ven tan claro. El propio Richard Stallman, fundador del proyecto GNU, ante la pregunta de qué es lo que debe hacer un desarrollador de software libre para ganar dinero, suele responder que puede trabajar como camarero.

4.4. Distribución geográfica

La obtención de datos geográficos de los desarrolladores es una cuestión que todavía ha de ser abordada de manera más científica. El problema que presentan los estudios cuyos resultados se están mostrando en este capítulo es que al tratarse de encuestas en Internet abiertas a todo aquel que quiera participar, la participación depende mucho de los sitios donde se haya anunciado, así como de la forma comose anunció. Para ser estrictos, cabe mencionar que las encuestas no buscaban representatividad en este sentido, sino más bien obtener la respuesta y/o la opinión del mayor número posible de desarrolladores de software libre.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 166]

Sin embargo, podemos aventurarnos a hacer unas cuantas afirmaciones al respecto, a sabiendas de que los datos no son tan fiables como los expuestos con anterioridad y que el margen de error es, por tanto, mucho más grande. Lo que parece un hecho constatable es que la gran mayoría de los desarrolladores de software libre provienen de países industrializados, siendo escasa la pre-

sencia de desarrolladores de países del llamado tercer mundo. No es de extrañar, por consiguiente, que el mapa de desarrolladores del proyecto Debian, por poner un ejemplo, concuerde con las fotografías de la Tierra de noche: allí donde hay luz –léase donde hay civilización industrializada– es donde suelen concentrarse en mayor medida los desarrolladores de software libre. Esto que en un principio podría parecer lógico, contrasta con las posibilidades potenciales que el software libre ofrece para países del tercer mundo.

Un claro ejemplo lo podemos encontrar en la siguiente tabla, que contiene los países de origen más frecuentes para los desarrolladores del proyecto Debian a lo largo de los últimos cuatro años. Se puede observar una tendencia a la descentralización del proyecto, algo que se constata en el hecho de que el crecimiento de los desarrolladores en Estados Unidos –el país que más aporta– es inferior a la media. Y es que, por lo general, los países han conseguido doblar el número de voluntarios en los últimos cuatro años, siendo Francia el ejemplo más claro en este sentido, ya que ha conseguido multiplicar por cinco su presencia. Considerando que los primeros pasos de Debian tuvieron lugar en el continente americano (en particular en Estados Unidos y Canadá), podemos ver que en los últimos cuatro años ha sufrido una *europeización* del proyecto. Suponemos que el siguiente paso será la ansiada mundialización con la incorporación de países sudamericanos, africanos y asiáticos (exceptuando Corea y Japón, ya bien representadas), aunque los datos que manejamos (dos desarrolladores en Egipto, China e India, uno en México, Turquía y Colombia en junio del 2003) no son muy halagüeños en este sentido.

Tabla 4-1. Países con mayor número de desarrolladores de Debian

País	1.7.1999	1.7.2000	1.7.2001	1.7.2002	20.6.2003
Estados Unidos	162	169	256	278	297
Alemania	54	58	101	121	136
Reino Unido	34	34	55	63	75
Australia	23	26	41	49	52
Francia	11	11	24	44	51

ANOTACIONES

País	1.7.1999	1.7.2000	1.7.2001	1.7.2002	20.6.2003
Canadá	20	22	41	47	49
España	10	11	25	31	34
Japón	15	15	27	33	33
Italia	9	9	22	26	31
Países Bajos	14	14	27	29	29
Suecia	13	13	20	24	27

Dentro del mundo del software libre (y no sólo en el caso de Debian), se mantiene una amplia discusión sobre la supremacía en el mundo del software libre entre Europa y Estados Unidos. Casi todos los estudios que se han venido realizando muestran que la presencia de desarrolladores europeos es ligeramente superior a la americana, efecto que queda mitigado por el hecho de que la población europea es mayor que la americana. Nos encontramos entonces ante una situación de guerra de cifras, ya que el número de desarrolladores per cápita favorece entonces a los norteamericanos, pero vuelve a ser favorable a los europeos si tenemos en cuenta en vez de las cifras de población absolutas solamente aquellas personas que cuentan con acceso a Internet.

En cuanto a países, las zonas con mayor implantación (en número de desarrolladores dividido por población) son las del norte de Europa (Finlandia, Suecia, Noruega, Dinamarca e Islandia) y centroeuropeas (Benelux, Alemania y Chequia), seguidos de Australia, Canadá, Nueva Zelanda y Estados Unidos. La zona mediterránea, a pesar de ser importante en magnitudes absolutas (debido a la gran población que tienen Francia, Italia y España), sin embargo, se encuentra por debajo de la media.

4.5. Dedicación

El número de horas que los desarrolladores de software libre dedican al software libre es uno de los aspectos más desconocidos. Nótese, además, que se trata de una de las grandes diferencias con el software generado por una empresa, donde tanto el equipo como la dedicación de cada miembro del equipo al desarrollo es conocido. El tiempo que los desarrolladores de software libre de-

ANOTACIONES

dican se puede tomar como una medida indirecta del grado de profesionalización. Antes de mostrar los datos de los que se dispone en la actualidad, es importante hacer notar que los datos han sido obtenidos de las estimaciones que los propios desarrolladores han proporcionado en varias encuestas, por lo que a la inexactitud inherente a este tipo de recolección de datos se ha de añadir un margen de error debido principalmente a lo que cada desarrollador entienda como tiempo de desarrollo. De esta forma, es seguro que muchos desarrolladores no cuentan el tiempo que dedican a leer el correo (o quizás sí), indicando sólo el tiempo que dedican a programar y a depurar. Por eso, todas las cifras que se muestren a continuación han de tomarse con el debido cuidado.

Los estudios que se han realizado hasta ahora muestran que como media cada desarrollador de software libre dedica en torno a 11 horas semanales. Sin embargo, esta cifra puede llevar rápidamente al engaño, ya que existe una gran varianza en la dedicación de los desarrolladores de software. En el estudio Rishab A. Ghosh; Ruediger Glott; Bernhard Krieger; Gregorio Robles (2002). *Free/libre and open source soft-ware: Survey and study - part iv: Survey of developers*, un 22,5% de los encuestados indicó que su aportación era inferior a las dos horas semanales, cifra que subía al 26,5% para los que dedicaban entre 2 y 5 horas semanales. Entre 6 y 10 horas es el tiempo que dedica un 21,0%, mientras que el 14,1% lo hacía entre 11 y 20 horas semanales. El 9,2% y el 7,1% de los encuestados afirmaban respectivamente que el tiempo que dedicaban a desarrollar software libre era entre 20 y 40 horas semanales y más de 40 horas semanales.

Lectura complementaria

[Ver bibliografía, apartados 122 y 104]

Tabla 4-2. Dedicación en horas semanales

Horas semanales	Porcentaje
Menos de 2 horas	22,5%
Entre 2 y 5 horas	26,1%
Entre 5 y 10 horas	21,0%
Entre 10 y 20 horas	14,1%
Entre 20 y 40 horas	9,2%
Más de 40 horas	7,1%

ANOTACIONES

Nota

Además de poder ver la profesionalización de los equipos de desarrollo de software libre, la dedicación en horas es un parámetro de gran importancia a la hora de poder realizar estimaciones de coste y hacer comparaciones con los modelos de desarrollo propietarios que se siguen en la industria. En el software libre, por ahora, sólo contamos con productos finales (nuevas entregas del software, sincronización de código nuevo en los sistemas de versiones) que no nos permiten conocer cuánto tiempo ha necesitado el desarrollador en conseguirlo.

El análisis de estas cifras nos muestra que en torno a un 80% de los desarrolladores realizan estas tareas en su tiempo libre, mientras que sólo uno de cada cinco podría considerarse como que dedica tanto tiempo a esta actividad como un profesional. Más adelante, en el capítulo de ingeniería del software podremos ver cómo este dato concuerda con la contribución de los desarrolladores, ya que ambas parecen seguir la ley de Pareto (ver apartado 7.6).

4.6. Motivaciones

Se ha especulado y se sigue especulando mucho sobre las motivaciones que hay detrás del desarrollo de software libre, sobre todo en su vertiente de actividad de tiempo libre (que, como hemos visto, corresponde a cerca de un 80% de los desarrolladores). Como en los apartados anteriores, sólo contamos con los datos de las encuestas, por lo que es importante darse cuenta de que se trata de lo que los desarrolladores responden, que puede ser más o menos coherente con la realidad. Los porcentajes que se van a mostrar a continuación superan en suma el 100%, ya que se daba la posibilidad a los encuestados de elegir varias respuestas.

En cualquier caso, de sus respuestas parece desprenderse que la mayoría quiere aprender y desarrollar nuevas habilidades (cerca de un 80%) y que muchos lo hacen para compartir conocimientos y habilidades (50%) o para participar en una nueva forma de cooperación (alrededor de un tercio). El primer dato no parece nada

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 72]

sorprendente, habida cuenta de que un profesional con mayores conocimientos se encuentra más cotizado que uno que no los posee. El segundo dato, sin embargo, no es tan fácil de explicar e incluso parece ir en contra de la afirmación de Nikolai Bezroukov, que viene a decir que los líderes de los proyectos de software libre tienen a buena cuenta no compartir toda la información de la que poseen para perpetuar su poder. Mientras tanto, la tercera opción más frecuente es, sin lugar a dudas, fiel reflejo de que los propios desarrolladores se muestran entusiasmados por la forma en la que generalmente se crea el software libre; es difícil encontrar una industria en la que un grupo de voluntarios levemente organizados pueda plantar cara tecnológicamente a las grandes compañías del sector.

Mientras la teoría *clásica* para explicar los motivos por los que los desarrolladores de software libre se dedican a aportar en proyectos de software libre gira en torno a la reputación y a beneficios económicos indirectos a medio y largo plazo, parece que los propios desarrolladores no están de acuerdo con estas afirmaciones. Sólo un 5% de los encuestados responde que desarrolla software libre para ganar dinero, mientras que el número de ellos que lo hacen por obtener reputación asciende a un 9%, lejos de las respuestas que se han presentado en el párrafo anterior. En cualquier caso, parece que el estudio de las motivaciones que tienen los desarrolladores para entrar a formar parte de la comunidad del software libre es una de las tareas primordiales con las que se han de enfrentar sociólogos y psicólogos en los próximos tiempos.

4.7. Liderazgo

Reputación y liderazgo son dos características con las que se ha tratado de explicar el éxito del software libre y en especial el del modelo de bazar, tal y como se verá en el capítulo dedicado a la ingeniería del software. Como pudimos ver en otro capítulo, el dedicado a las licencias del software, existen ciertas diferencias entre las licencias de software libre y sus homólogas en el campo de la documentación. Esas diferencias radican en la forma como se preservan la autoría y la opinión del autor –más acentuada en textos que en programas.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 104]

En Rishab A. Ghosh; Ruediger Glott; Bernhard Krieger; Gregorio Robles (2002). *Free/libre and open source soft-ware: Survey and study - part iv: Survey of developers*, se incluyó una pregunta en la que se instaba a los desarrolladores a indicar qué personas de una lista dada le eran conocidas, no necesariamente e manera personal. Los resultados se muestran en la tabla 4-3. Éstos determinan que se pueden aglutinar a estas personas en tres grupos claramente diferenciados:

Tabla 4-3. Grado de conocimiento de desarrolladores importantes

Desarrollador	Conocido por
Linus Torvalds	96,5%
Richard Stallman	93,3%
Miguel de Icaza	82,1%
Eric Raymond	81,1%
Bruce Perens	57,7%
Jamie Zawinski	35,8%
Mathias Ettrich	34,2%
Jörg Schilling	21,5%
Marco Presenti Gritti	5,7%
Bryan Andrews	5,6%
Guenter Bartsch	3,5%
Arpad Gereoffy	3,3%
Martin Hoffstede	2,9%
Angelo Roulini	2,6%
Sal Valliger	1,2%

Un primer grupo de gente con claras connotaciones filosófico-históricas dentro del mundo de software libre (aunque cuenten, como se puede ver, también con aptitudes técnicas notables):

Lectura complementaria

[Ver bibliografía, apartado 195]

Lectura complementaria

[Ver bibliografía, apartados 185, 184, 187 y 95]

- **Linus Torvalds**: creador del núcleo Linux, el kernel del sistema operativo más utilizado. Coautor de *Just For Fun: the story of an accidental revolutionary*.
- **Richard Stallman**: ideólogo y fundador de la Fundación del Software Libre y desarrollador en varios proyectos GNU. Autor de varios escritos muy importantes dentro del mundo del software libre

ANOTACIONES

- **Miguel de Icaza**: cofundador del proyecto GNOME y de Ximian Inc. Desarrollador de parte de GNOME y de MONO

- **Eric Raymond**: impulsor de la Open Source Initiative, autor de "La catedral y el bazar". Desarrollador principal de fetchmail.

- **Bruce Perens**: antiguo líder del proyecto Debian. Impulsor (converso) de la Open Source Initiative. Desarrollador de la herramienta e-fence

- **Jamie Zawinsky**: ex-desarrollador de Mozilla, famoso por una carta en 1999 en la que dejaba el proyecto Mozilla argumentando que el modelo utilizado no iba a dar frutos nunca.

- **Mathias Ettrich**: fundador de KDE. Desarrollador de LyX y otros.

Lectura complementaria

[Ver bliografía, apartado 214]

En el segundo grupo tenemos a desarrolladores. Para esta encuesta se tomaron los nombres de los desarrolladores principales de los seis proyectos más populares en el índice de aplicaciones de software libre FreshMeat. Se puede ver que (a excepción de Linus Torvalds, por motivos obvios, y de Jörg Schilling), el grado de conocimiento de estos desarrolladores es pequeño.

- Jörg Schilling, creador de cdrecord entre otras aplicaciones
- Marco Presenti Gritti, desarrollador principal de Galeon
- Bryan Andrews, desarrollador de Apache Toolbox
- Guenther Bartsch, creador de xine
- Arpad Gereoffy, desarrollador de MPEGplayer

El tercer grupo lo componen los nombres de las últimas tres *personas.*Estos nombres fueron inventados por el equipo de la encuesta para poder comprobar el margen de error de las respuestas.

De los resultados se pueden observar dos cosas: lprimera es que el margen de error de las respuestas se puede considerar pequeño (menor a un 3%). Y la segunda es que la mayoría de los desarrolladores de las aplicaciones de software libre más populares son tan conocidos como personas que no existen. Este dato puede dar que pensar a los que aducen el hecho de buscar fama como una de las primeras causas para desarrollar software libre.

ANOTACIONES

4.8. Resumen y conclusiones

Este capítulo ha pretendido arrojar un poco de luz sobre el ampliamente desconocido tema de la gente que dedica su tiempo al software libre. En líneas generales se puede afirmar que el desarrollador de software libre es un varón joven con estudios universitarios (o en vías de conseguirlos). La relación del mundo del software libre con la universidad (estudiantes y profesores) es muy estrecha, aunque sigue predominando el desarrollador que no tiene que ver nada con el ámbito académico.

En cuanto a la dedicación en número de horas, se ha mostrado cómo existe una gran desigualdad al estilo de la postulada en la ley de Pareto. Las motivaciones de los desarrolladores –según ellos mismos– lejos de ser monetarias y egocéntricas, tal y como suelen asumir economistas y psicólogos, está más bien centrada en compartir y aprender. Para finalizar, se ha mostrado un cuadro de los personajes del mundo del software libre más significantes (y otros no tanto, como se ha podido ver) y se ha demostrado que la reputación en la gran comunidad del software libre suele depender de más razones que solamente de la codificación de una aplicación libre exitosa.

ANOTACIONES

5. Economía

En este capítulo se tratan algunos aspectos económicos relacionados con el software libre. Se comienza mostrando cómo se financian los proyectos de software libre (cuando efectivamente se financian, ya que en muchos casos se desarrollan únicamente con trabajo y recursos aportados voluntariamente). A continuación, se exponen los principales modelos de negocio que están poniendo en práctica las empresas relacionadas directamente con el software libre. El capítulo termina con un pequeño estudio sobre la relación entre el software libre y los monopolios en la industria del software.

5.1. Financiación de proyectos de software libre

El software libre se desarrolla de muchas formas distintas, y con mecanismos para conseguir recursos que varían muchísimo de un caso a otro. Cada proyecto libre tiene su propia forma de financiarse, desde el que está formado completamente por desarrolladores voluntarios y utiliza solamente recursos cedidos altruistamente, hasta el que es llevado a cabo por una empresa que factura el 100% de sus costes a una entidad interesada en el desarrollo correspondiente.

En este apartado nos vamos a centrar en los proyectos donde hay financiación externa, y no todo el trabajo realizado es voluntario. En estos casos, hay algún tipo de flujo de capital con origen externo al proyecto que se encarga de aportar recursos para su desarrollo. De esta manera, el software libre construido puede considerarse, de alguna manera, como un producto de esta financiación externa. Por ello, es común que sea esa fuente externa quien decide (al menos parcialmente) cómo y en qué se gastan los recursos.

En cierto sentido, esta financiación externa para proyectos libres puede considerarse como un tipo de patrocinio, aunque este patrocino no tiene por qué ser desinteresado (y habitualmente no lo es). En los siguientes apartados se comentan los tipos de financiación externa más habituales. Mientras el lector se dedique a ellos, conviene, no

ANOTACIONES

obstante, no olvidar que ésta es sólo una de las maneras que tienen los proyectos que construyen software libre de conseguir recursos. Hay otras, y entre ellas la más importante: el trabajo de muchos desarrolladores voluntarios (como se discute en el capítulo 4).

5.1.1. Financiación pública

Un tipo muy especial de financiación de proyectos libres es la pública. La entidad financiadora puede ser directamente un gobierno (local, regional, nacional o incluso supranacional) o una institución pública (por ejemplo, una fundación). En estos casos, la financiación suele ser similar a la de los proyectos de investigación y desarrollo, y de hecho es muy habitual que provenga de entidades públicas promotoras de I+D. Normalmente, la entidad financiadora no busca recuperar la inversión (o al menos no de forma directa), aunque suele tener objetivos claros (favorecer la creación de tejido industrial e investigador, promover cierta tecnología o cierto tipo de aplicaciones, etc.).

En la mayor parte de estos casos, no se encuentra explícitamente la financiación de productos o servicios relacionados con software libre, sino que suelen ser el subproducto de un contrato con otros fines más generales. Por ejemplo, la Comisión Europea, dentro de sus programas de investigación, financia proyectos orientados a mejorar la competitividad europea en ciertas áreas. Algunos de estos proyectos tienen como parte de sus objetivos usar, mejorar y crear software libre en su ámbito de investigación (como herramienta para la investigación, o como producto derivado de ella).

Las motivaciones para este tipo de financiación son muy variadas, pero pueden destacarse las siguientes:

- Científica

Éste es el caso más habitual en proyectos de investigación financiados con fondos públicos. Aunque su objetivo no es producir software, sino investigar en un determinado campo (relacionado o no con la informática), es muy posible que para ello sea preciso desarrollar programas que se usen como herramientas necesarias

ANOTACIONES

para alcanzar las metas del proyecto. Normalmente, el proyecto no está interesado en comercializar estas herramientas, o incluso está activamente interesado en que otros grupos las utilicen y mejoren. En estos casos, es bastante habitual distribuirlos como software libre. Cabe mencionar que los recursos que consiguió el grupo que realiza la investigación se dedicaron en parte a la producción de este software, por lo que se puede decir que fue desarrollado con financiación pública.

- Promoción de estándares

Tener una implementación de referencia es una de las mejores formas de promover un estándar. En muchos casos eso supone tener programas que formen parte de esa implementación (o si el estándar se refiere al campo del software, que sean la implementación ellos mismos). Para que la implementación de referencia sea de utilidad en la promoción del estándar, es preciso que esté disponible, al menos para comprobar su interoperabilidad entre todos los que quieran desarrollar productos que se acojan a ese estándar. En muchos casos es conveniente también que los fabricantes puedan directamente adaptar la implementación de referencia para usarla en sus productos, si así lo desean. De esta manera se desarrollaron, por ejemplo, muchos de los protocolos de Internet que hoy se han convertido en norma universal. En estos casos, la liberación de esas implementaciones de referencia como software libre puede ayudar mucho en esa promoción, ya que el software libre es un subproducto de la promoción del estándar, y habitualmente, quien se encarga de esta promoción es una entidad pública (aunque a veces es un consorcio privado).

- Social

El software libre es una herramienta de gran interés en la creación de la infraestructura básica para la sociedad de la información. Las entidades interesadas en utilizar software libre para ayudar al acceso universal a esta sociedad de la información pueden financiar proyectos relacionados con el software libre (normalmente proyectos de desarrollo de nuevas aplicaciones o de adaptación de otras existentes).

ANOTACIONES

Nota

Un ejemplo de financiación pública con finalidad fundamentalmente social es el caso de LinEx, promovido por la Junta de Extremadura (Extremadura, España) para promover la sociedad de la información fundamentalmente en lo que a alfabetización informática se refiere. La Junta ha financiado el desarrollo de una distribución basada en Debian para conseguir este fin. Otro caso similar es el de la financiación por parte del Gobierno alemán de desarrollos de GnuPG orientados a facilitar su uso a los usuarios no experimentados, con la idea de favorecer el uso del correo seguro entre sus ciudadanos.

Ejemplo

El desarrollo de GNAT

Un caso notorio de financiación pública para el desarrollo de software libre fue el del compilador GNAT. GNAT, compilador de Ada, fue financiado por el proyecto Ada9X del Departamento de Defensa de EE.UU., con la idea de disponer de un compilador de la nueva versión del lenguaje de programación Ada (la que luego fue Ada95), cuyo uso trataba de promover en aquella época. Una de las causas que se habían identificado en cuanto a la adopción de la primera versión de Ada (Ada83) por las empresas de software era la tardía disposición de un compilador del lenguaje, y su alto precio cuando estuvo finalmente disponible. Por ello, trataron de que no ocurriese lo mismo con Ada95, asegurándose de que hubiera un compilador de forma prácticamente simultánea con la publicación del nuevo estándar del lenguaje.

Para lograrlo, el proyecto Ada9X contrató un proyecto con un equipo de la Universidad de Nueva York (NYU), por un importe aproximado de 1 millón de USD, para la realización de una "prueba de concepto" de compilador de Ada95. Con estos fondos, y aprovechando la existencia de GCC (el compilador de C de GNU, del que se aprovechó la mayor parte del dorsal), el equipo de NYU construyó efectivamente el primer compilador

ANOTACIONES

de Ada95, que liberó bajo la GNU GPL. El compilador tuvo tanto éxito que al finalizar el proyecto parte de sus constructores fundaron una empresa (Ada Core Tecnologies), que desde entonces se ha convertido en líder en el mercado de compiladores y herramientas de ayuda a la construcción de programas en Ada.

Es notable cómo en este proyecto se puede ver la combinación de elementos de investigación (este proyecto avanzó en el conocimiento sobre la construcción de frontales y de sistemas de tiempo de ejecución para compiladores de lenguajes tipo Ada) y de promoción de estándares (que era el objetivo más claro de su financiador).

5.1.2. Financiación privada sin ánimo de lucro

Éste es un tipo de financiación con muchas características similares a las del caso anterior, donde la financiación la realizan normalmente fundaciones u organizaciones no gubernamentales. La motivación directa en estos casos suele ser producir software libre para su uso en algún ámbito que la entidad financiadora considera especialmente relevante, pero también puede encontrarse la motivación indirecta de contribuir a resolver un problema (por ejemplo, una fundación dedicada a promover la investigación sobre una enfermedad puede financiar la construcción de un programa estadístico que ayude al análisis de grupos de experimentación donde se está estudiando esa enfermedad).

En general, tanto los motivos para realizar esta financiación como sus mecanismos son muy similares a los de financiación pública, aunque naturalmente están siempre marcados por los objetivos de la entidad que financia.

Nota

Probablemente el caso paradigmático de fundación que promueve el desarrollo de software libre sea la Free Software Foundation (FSF). Desde mediados de la década de 1980 esta fundación se dedica a la promoción del proyecto GNU, y a fomentar en general el desarrollo del software libre.

ANOTACIONES

Otro caso interesante, aunque en otro ámbito bastante diferente, es la Open Bioinformatics Foundation. Entre los fines de esta fundación se encuentra la de promover el desarrollo de los programas informáticos que son básicos para investigación en cualquiera de las ramas de la bioinformática. Y en general, realiza esta promoción financiando y ayudando a la construcción de programas libres.

5.1.3. Financiación por quien necesita mejoras

Otro tipo de financiación para el desarrollo de software libre, ya no tan altruista, es el que tiene lugar cuando alguien necesita mejoras en un producto libre. Por ejemplo, una empresa necesita para uso interno cierta funcionalidad en un programa dado. O bien necesita que ciertos errores en un programa sean corregidos. En este tipo de casos, lo habitual es que la empresa en cuestión contrate el desarrollo que necesita. Este desarrollo muy habitualmente (bien porque lo impone la licencia del programa modificado, bien porque la empresa lo decide así) es software libre.

Ejemplo

El caso de Corel y Wine

A finales de la década de 1990, Corel decidió portar sus productos a GNU/Linux. En el proceso de realizarlo, descubrió que un programa libre diseñado para facilitar la ejecución de binarios para Windows en entornos Linux podría permitirle muchos ahorros de desarrollo. Pero para ello era preciso mejorarlo, fundamentalmente añadiéndole la emulación de cierta funcionalidad de Windows que usaban los programas de Corel.

Para que se realizaran estas mejoras, Corel contrató a Macadamian, que contribuyó con sus mejoras al proyecto Wine. Con ello, tanto Corel como Wine salieron beneficiados.

ANOTACIONES

5.1.4. Financiación con beneficios relacionados

En este caso, lo que busca la entidad financiadora es conseguir beneficios en productos relacionados con el programa a cuyo desarrollo aporta recursos. Normalmente, en estos casos los beneficios que obtiene la empresa financiadora no son exclusivos, ya que otras pueden entrar también en el mercado de la venta de productos relacionados. Pero, o bien la cuota de mercado que tiene es suficiente como para que no le preocupe mucho repartir la tarta con otros, o bien tiene alguna ventaja competitiva clara.

Algunos ejemplos de productos relacionados con un software dado son:

- Libros

La empresa en cuestión vende manuales, guías de uso, textos para cursos, etc. relacionados con el programa libre que ayuda a financiar. Por supuesto otras empresas pueden vender también libros relacionados con aquél, pero normalmente el hecho de financiar el proyecto le proporcionará acceso a desarrolladores clave antes que a sus competidores, o simplemente conseguirá con ello una buena imagen de cara a la comunidad de usuarios del programa en cuestión.

- Hardware

Si una empresa financia el desarrollo de sistemas libres para cierto tipo de hardware, puede dedicarse a vender con más facilidad ese tipo de hardware. De nuevo, al ser libre el software desarrollado, pueden aparecer competidores que vendan aparatos del mismo tipo, usando esos desarrollos, pero sin colaborar a su financiación. Aun así, la empresa en cuestión dispone de varias ventajas sobre sus competidores, y una de ellas puede ser que su posición como aportadora de recursos para el proyecto le permite influir para conseguir que los desarrollos que se realicen con prioridad sean los que más le interesen.

- CD con programas

Probablemente el modelo de este tipo más conocido es el de las empresas que financian ciertos desarrollos que luego aplican a su dis-

ANOTACIONES

tribución de software. Por ejemplo, tener un buen entorno de escritorio puede ayudar mucho a vender CD con una cierta distribución de GNU/Linux, y por tanto financiar su desarrollo puede ser un buen negocio para quien los vende.

Es importante darse cuenta de que, para estar en este apartado, la financiación en cuestión ha de hacerse con ánimo de lucro, y para ello la entidad financiadora debe percibir algún beneficio posible en esta financiación. En los casos reales, sin embargo, es habitual que haya siempre una combinación de ánimo de lucro y altruismo cuando una empresa aporta recursos para que se realice un programa libre del cual espera beneficiarse indirectamente.

Nota

Un caso muy conocido de aporte de recursos a un proyecto, si bien de forma relativamente indirecta, es la ayuda que la editorial O'Reilly presta al desarrollo de Perl. Naturalmente, no es casualidad que esa editorial sea también una de las principales editoras sobre temas relacionados con Perl. En cualquier caso, es obvio que O'Reilly no tiene la exclusiva de la edición de libros de ese tipo, y otras editoriales compiten en ese segmento de mercado, con diverso éxito.

VA Software (en sus comienzos VA Research, y más tarde VA Linux) ha colaborado activamente en el desarrollo del kernel de Linux. Con ello ha conseguido, entre otras cosas, colaborar a que su su continuidad esté asegurada, lo que era especialmente crítico para ella, de cara a sus clientes, cuando su principal negocio era vender equipos con GNU/Linux preinstalado.

Red Hat ha financiado el desarrollo de muchos componentes de GNOME, con lo que ha conseguido fundamentalmente tener un entorno de escritorio para su distribución, lo que ha contribuido a aumentar sus ventas. Como en otros casos, otros fabricantes de distribuciones se han beneficiado de este desarrollo, aunque muchos de ellos no hayan colaborado con el proyecto GNOME en la misma medida que Red Hat (y

ANOTACIONES

no son pocos los que no han colaborado en nada). A pesar de ello, Red Hat se beneficia de su contribución a GNOME.

5.1.5. Financiación como inversión interna

Hay empresas que, directamente como parte de su modelo de negocio, desarrollan software libre. Por ejemplo, una empresa puede decidir iniciar un nuevo proyecto libre en un ámbito donde percibe que puede haber oportunidades de negocio, con la idea de rentabilizar posteriormente esta inversión. Este modelo podría considerarse una variante del anterior (financiación indirecta), siendo los "beneficios relacionados" las ventajas que obtenga la empresa por la producción del programa libre. Pero, por ser en este caso el producto libre en sí mismo el que se espera que produzca los beneficios, parece conveniente abrir una clasificación específica para estos casos.

Este tipo de financiación da lugar a varios modelos de negocio. Cuando se analicen éstos –en el apartado 5.2– se explicarán también las ventajas que normalmente obtiene una empresa de esta inversión en un proyecto, y qué métodos suelen utilizarse para rentabilizarla. De todos modos, cabe destacar que en algunos casos puede que el software en cuestión se desarrolle simplemente para satisfacer las necesidades de la propia empresa, y que sólo después se decida liberar, y quizás abrir una línea de negocio relacionada con él.

Nota

Digital Creations (hoy Zope Corporation) es uno de los casos más conocidos de empresa que se dedica al desarrollo de software libre con la esperanza de rentabilizar su inversión. El proyecto libre en que más está invirtiendo es Zope, un servidor de aplicaciones que está teniendo un cierto éxito. Su historia con el software libre comenzó cuando la entonces Digital Creations buscaba capital-riesgo para desarrollar su servidor de aplicaciones propietario, hacia 1998. Uno de los grupos interesados en invertir en ellos (Opticality Ventures) les puso como condición que el producto resultante debía ser libre, porque

ANOTACIONES

ANOTACIONES

Nota

en caso contrario no veían cómo iba a poder conseguir una cuota de mercado significativa. Digital Creations se decidió por ese camino, y pocos meses después anunciaba la primera versión de Zope (unos años después cambió su nombre). Hoy día Zope Corporation está especializada en ofrecer servicios de consultoría, formación y soporte para sistemas de gestión de contenidos basados en Zope, y otros productos en los que sin duda Zope es la piedra angular.

Ximian (antes Helix Code) es un caso bien conocido de desarrollo de aplicaciones libres en entorno empresarial. Muy ligada desde sus orígenes al proyecto GNOME, Ximian ha producido sistemas software como Evolution (un gestor de información personal que tiene una funcionalidad relativamente similar a la ofrecida por MS Outlook), Red Carpet (un sistema fácil de usar para la gestión de paquetería en un sistema operativo) y Mono (una implementación de gran parte de .NET). La empresa fue fundada en octubre de 1999, y atrajo a muchos desarrolladores de GNOME que pasaron a formar parte de sus desarrolladores (en muchos casos, continuando su colaboración con el proyecto GNOME). Ximian se posicionó como una empresa de ingeniería experta en adaptaciones de GNOME, en la construcción de aplicaciones basadas en GNOME, y en general en proporcionar servicios de desarrollo basados en software libre, especialmente de herramientas relacionadas con el entorno de escritorio. En agosto del 2003, Ximian fue adquirida por Novell.

Lectura complementaria

[Ver bliografía, apartado 16]

Cisco Enterprise Print System (CEPS) es un sistema de gestión de impresión para organizaciones con gran cantidad de impresoras. Fue desarrollado internamente en Cisco, para satisfacer sus propias necesidades, y liberado en el año 2000 bajo la GNU GPL. Es difícil estar seguro de los motivos que tuvo Cisco para hacer esto, pero es posible que tuviera que ver con la búsqueda de contribuciones externas (informes de

error, nuevos controladores, parches, etc.). En cualquier caso, lo que está claro es que, al no tener Cisco ningún plan para comercializar el producto y no estar muy claro su mercado potencial, no tenía mucho que perder con esta decisión.

5.1.6. Otros modos de financiación

Hay otros modos de financiación difíciles de clasificar entre los anteriores. Entre ellos pueden destacarse, a modo de ejemplo, los siguientes:

- Utilización de mercados para poner en contacto a desarrolladores y clientes

La idea que sostiene este modo de financiación es que, sobre todo para pequeños desarrollos, es difícil que un cliente que lo desea pueda entrar en contacto con un desarrollador que pueda acometerlo de forma eficiente. Para mejorar esta situación, se postulan los mercados de desarrollo de software libre, donde los desarrolladores publicarían sus habilidades y los clientes los desarrollos que precisan. Una vez un desarrollador y un cliente se han puesto de acuerdo, tenemos una situación similar a la ya descrita como "financiación por quien necesita las mejoras" en el apartado 5.1.3.

Ejemplo

SourceXchange

SourceXchange fue un ejemplo de mercado para poner en contacto a desarrolladores con sus clientes potenciales. Para ofrecer un proyecto, un cliente escribía una RFP (*request for proposal*, o 'petición de propuesta'), especificando qué desarrollo se necesita y cuántos recursos se está dispuesto a proporcionar para ese desarrollo. Estas RFP se publicaban en el sitio. Cuando un desarrollador veía una que le interesaba, hacía una oferta para ella. Cuando un desarrollador y un cliente se ponían de acuerdo en los términos del desarrollo,

ANOTACIONES

comenzaba un proyecto. Normalmente, cada proyecto estaba supervisado por un *peer reviewer*, un revisor, que se encargaba de asegurarse de que el desarrollador cumplía las especificaciones, que efectivamente éstas tenían sentido, que aconsejaba sobre cómo llevar adelante el proyecto, etc. SourceXchange (propiedad de la empresa CollabNet) se encargaba de ofrecer el sitio, garantizar la competencia de los revisores, asegurarse del pago en caso de que los proyectos se completasen, y ofrecer herramientas para el seguimiento del proyecto (servicios que facturaba al cliente). El primer proyecto mediado por SourceXchange se terminó en marzo del 2000, pero poco más de un año después, en abril del 2001, el sitio cerró.

- Venta de bonos para financiar un proyecto

Esta idea de financiación es similar a la de los mercados de bonos a los que acuden las empresas, pero orientado al desarrollo de software libre. Presenta unas cuantas variantes, pero una de las más conocidas funciona como sigue. Cuando un desarrollador (individual o empresa) tiene idea de un nuevo programa o de una mejora a uno existente, lo escribe en forma de especificación, estima el coste que tendría su desarrollo y emite bonos para su construcción. Estos bonos tienen un valor que se ejecuta sólo si el proyecto finalmente se termina. Cuando el desarrollador ha vendido suficientes bonos, comienza el desarrollo, que va financiando con préstamos basados en ´éstos. Cuando termina el desarrollo y se certifica por alguna tercera parte independiente que efectivamente lo realizado cumple con las especificaciones, el desarrollador "ejecuta" los bonos que había vendido, paga sus deudas, y lo que le queda son los beneficios que obtiene por el desarrollo.

¿Quién estaría interesado en adquirir los mencionados bonos? Obviamente, los usuarios que desearan que ese nuevo programa o esa mejora a uno ya existente se realizaran. De alguna manera, este sistema de bonos permitiría que las partes interesadas fijaran (siquiera parcialmente) las prioridades de los desarrolladores mediante la compra de bonos. Ello permitiría al mismo tiempo que no hiciese fal-

ANOTACIONES

ta que una sola entidad asumiese los costes de desarrollo, sino que podrían repartirse entre muchas (incluyendo individuos), que además sólo tendrían que pagar si finalmente el proyecto termina con éxito. Un mecanismo muy similar a éste se propone, con mucho más detalle, en Chris Rasch. (2001). "The Wall Street performer protocol. using software completion bonds to fund open source software development". *First Monday*, (vol. 6, núm. 6, mayo).

Nota

El sistema de bonos descrito está basado en el *Street Performer Protocol* (protocolo del artista callejero), un mecanismo basado en el comercio electrónico diseñado para facilitar la financiación privada de trabajos de creación libres. Resumiendo, quien esté interesado en que se realice un determinado trabajo prometerá formalmente pagar una cierta cantidad si el trabajo se realiza y es publicado libremente. Sus intenciones son buscar una nueva manera de financiar trabajos relativamente pequeños que queden a disposición de todo el mundo, pero puede extenderse de muchas formas (los bonos para la construcción de software libre son una de ellas). Puede verse un pequeño caso de puesta en práctica de un derivado de este protocolo (el *Rational Street Performer Protocol*, Protocolo Racional del Artista Callejero, Paul Harrison en http://thecircle.org.au/funding_results.html, donde se aplica a la consecución de fondos para la financiación de parte de The Circle, un proyecto de software libre.

Lectura complementaria

[Ver bibliografía, apartados 132, 133 y 116]

- Cooperativas de desarrolladores

En este caso, los desarrolladores de software libre, en lugar de trabajar individualmente o para una empresa, se reúnen en algún tipo de asociación (normalmente similar a una cooperativa). Por lo demás, su funcionamiento es similar al de una empresa, matizado quizás por su compromiso ético con el software libre, que puede ser parte de sus estatutos (aunque también una empresa puede hacer esto). En este tipo de organizaciones pueden darse combinaciones variadas de trabajo voluntario con trabajo remunerado. Un ejemplo de este tipo de organización es Free Developers.

ANOTACIONES

5.2. Modelos de negocio basados en software libre

Además de los mecanismos de financiación de los proyectos, ya tratados, otro aspecto muy relacionado con la economía que merece la pena tratar es el de los modelos de negocio. Ya al hablar de estos mecanismos de financiación se han mencionado algunos brevemente, ahora en este apartado los describiremos de forma algo más metódica.

En general, puede decirse que son muchos los modelos de negocio que se están explorando en torno al software libre, algunos más clásicos y otros más innovadores. Hay que tener en cuenta que entre los modelos más habituales en la industria del software no es fácil usar aquellos basados en la venta de licencias de uso de programas producidos, pues en el mundo del software libre ése es un mecanismo de financiación muy difícil de explotar. Sin embargo, sí que se pueden utilizar los que están basados en el servicio a terceros, con la ventaja de que, sin ser necesariamente el productor de un programa, puede darse soporte completo sobre él.



Venta de software libre a tanto por copia

En el mundo del software libre es difícil cobrar licencias de uso, pero en general no imposible. Normalmente, no hay nada en las definiciones de software libre que impida que una empresa cree un producto y sólo se lo distribuya a quien le pague una cierta cantidad. Por ejemplo, un determinado productor podría decidir distribuir su producto con una licencia libre, pero sólo a quien le pague 1.000 euros por copia (de forma similar a como se hace en el mundo clásico del software propietario).

Sin embargo, aunque esto es teóricamente posible, en la práctica es bastante difícil de que suceda. Porque una vez que el productor ha *vendido* la primera copia, quien la recibe puede estar motivado a tratar de recu-

ANOTACIONES

perar su inversión vendiendo más copias a menor precio (algo que no puede prohibir la licencia del programa, si éste es libre). En el ejemplo anterior, podría tratar de vender 10 copias a 100 euros cada una, con lo que el producto le saldría gratis (y además dificultaría mucho que el productor original vendiese otra copia a 1.000 euros, si el producto puede obtenerse legalmente por la décima parte). Es fácil deducir cómo este proceso continuaría en cascada hasta la venta de copias a un precio cercano al coste marginal de copia, que con las tecnologías actuales es prácticamente cero.

Aun así, y teniendo en cuenta que el mecanismo descrito propiciará que normalmente el productor no pueda poner un precio (especialmente un precio alto) al simple hecho de la redistribución del programa, hay modelos de negocio que implícitamente hacen justamente eso. Un ejemplo es el de las distribuciones de GNU/Linux, que se venden por un precio muy bajo comparado con sus competidores propietarios, pero superior (y normalmente claramente superior) al coste de copia (incluso cuando se pueden bajar libremente de Internet). Por supuesto, en estos casos juegan además otros factores, como la imagen de marca o la comodidad para el consumidor. Pero no es éste el único caso. Por lo tanto, más que indicar que con software libre *no se puede vender a tanto por copia*, hay que tener en cuenta que es más difícil de hacer, y probablemente se obtendrá menos beneficio, pero que puede haber modelos basados justamente en eso.

Dadas estas limitaciones (y estas ventajas) desde hace unos años se están probando variantes de los modelos de negocio habituales en la industria que son viables con software, a la vez que se buscan modelos innovadores, que tratan de explotar las nuevas posibilidades que ofrece el software libre. Sin duda en los próximos años veremos aún más experimentación en este campo, y también tendremos más información sobre qué modelos pueden funcionar bien, y en qué circunstancias.

ANOTACIONES

En este apartado vamos a ofrecer una panorámica de los que más habitualmente nos encontramos hoy día, agrupados con la intención de mostrar al lector lo que tienen en común y lo que les diferencia, centrándonos en los modelos basados en el desarrollo y los servicios en torno a un producto de software libre. Los ingresos en este caso provienen directamente de estas actividades de desarrollo y servicios para el producto, pero no necesariamente implican desarrollo de nuevos productos. Cuando sí que se hace desarrollo, estos modelos tienen como *subproducto* la financiación de productos de software libre, por lo que son modelos especialmente interesantes, y su impacto sobre el mundo del software libre en general puede ser grande.

En cualquier caso, y aunque aquí ofrecemos una clasificación relativamente estricta, no hay que olvidar que casi todas las empresas usan en realidad modelos mixtos, que son combinaciones de los modelos que hemos descrito, entre sí y con otros más tradicionales.

5.2.1. Mejor conocimiento

La empresa que utiliza este modelo de negocio trata de rentabilizar su conocimiento de un producto (o conjunto de productos) libres. Sus ingresos provendrán de clientes a los que venderá servicios relacionados con ese conocimiento, típicamente desarrollos basados en el producto, modificaciones, adaptaciones, instalaciones e integraciones con otros. La ventaja competitiva de la empresa estará en gran medida relacionada con el mejor conocimiento del producto en cuestión que el que tienen sus competidores: por ello, la empresa estará especialmente bien situada si es productora del producto o participa en el proyecto que lo produce.

Ésta es una de las razones por la que las empresas que utilizan este modelo suelen participar activamente en los proyectos relacionados con el software sobre el que tratan de vender servicios: es una forma muy eficiente de obtener conocimiento sobre él, y lo que es más importante, de que ese conocimiento sea reconocido. Desde luego, explicarle a un cliente que entre los empleados hay varios desarrolladores del proyecto que produce el software, que, por ejemplo, se quiere modificar, puede ser una buena garantía.

ANOTACIONES



Relación con los proyectos de desarrollo

Por lo tanto, este tipo de empresas tiene un gran interés en dar la imagen de que poseen un buen conocimiento de determinados productos libres. Una interesante consecuencia de ello es que su apoyo a proyectos de software libre (por ejemplo, participando activamente en ellos, o permitiendo a sus empleados que lo hagan durante su jornada laboral) no es por lo tanto algo meramente filantrópico. Por el contrario, puede ser uno de los activos más rentables de la empresa, ya que sus clientes lo valorarán muy positivamente como una muestra clara de que conocen el producto en cuestión. Además, de esta forma podrán seguir muy de cerca el desarrollo, tratando de asegurarse, por ejemplo, de que algunas de las mejoras demandadas por sus clientes pasan a formar parte del producto desarrollado por el proyecto.

Analizándolo desde un punto de vista más general, ésta es una situación donde ambas partes, la empresa y el proyecto de desarrollo, ganan con la colaboración que se lleva a cabo. El proyecto gana por el desarrollo realizado por la empresa, o porque algunos de sus desarrolladores pasan a ser remunerados (siquiera parcialmente) por su trabajo en el proyecto. La empresa gana en conocimiento del producto, en imagen hacia sus clientes y en una cierta influencia sobre el proyecto.

Los servicios que proporcionan este tipo de empresas pueden ser muy amplios, pero normalmente consisten en desarrollos a medida, adaptaciones o integraciones de los productos en los que son expertas, o bien servicios de consultoría, donde aconsejan a sus clientes cómo utilizar mejor el producto en cuestión (especialmente si es complejo, o si su correcto funcionamiento es crítico para el cliente en cuestión).

ANOTACIONES

Ejemplo

Algunos ejemplos de empresas que hasta cierto punto utilizan este modelo de negocio son los siguientes:

- LinuxCare

Fundada en 1996, proporcionaba en sus orígenes servicios de consultoría y soporte para GNU/Linux y software libre en EE.UU., y su plantilla estaba compuesta fundamentalmente por expertos en GNU/Linux. Sin embargo, en 1992 cambió sus objetivos, y desde entonces se ha especializado en proporcionar servicios casi exclusivamente a Linux ejecutando sobre máquinas virtuales z/VM en grandes ordenadores de IBM. Su modelo de negocio ha cambiado también a *mejor conocimiento con limitaciones*, al ofrecer como parte fundamental de sus servicios una aplicación no libre, Levanta.

Lectura complementaria

[Ver bibliografía, apartado 84]

- Alcove

Fundada en 1997 en Francia, proporciona fundamentalmente servicios de consultoría, consultoría estratégica, soporte y desarrollo para software libre. Desde su fundación, Alcove ha mantenido en plantilla a desarrolladores de varios proyectos libres, y ha tratado de rentabilizarlo en términos de imagen. También ha tratado de ofrecer una imagen, en general, de empresa vinculada a la comunidad de software libre, por ejemplo, colaborando con asociaciones de usuarios y dando publicidad a sus colaboraciones con proyectos libres (por ejemplo, desde Alcove-Labs).

Lectura complementaria

[Ver bibliografía, apartados 4 y 5]

5.2.2. Mejor conocimiento con limitaciones

Estos modelos son similares a los expuestos en el apartado anterior, pero tratando de limitar la competencia a que pueden verse sometidos. Mientras que en los modelos *puros* basados en el mejor conocimiento cualquiera puede, en principio, entrar en competencia, ya

ANOTACIONES

que el software utilizado es el mismo (y libre), en este caso se trata de evitar esa situación poniendo barreras a esa competencia. Estas barreras suelen consistir en patentes o licencias propietarias, que normalmente afectan a una parte pequeña (pero fundamental) del producto desarrollado. Por esta razón, estos modelos pueden considerarse en realidad como mixtos, en el sentido de que están a caballo entre el software libre y el propietario.

En muchos casos, la comunidad del software libre desarrolla su propia versión para ese componente, con lo que la ventaja competitiva puede desaparecer, o incluso volverse en contra de la empresa en cuestión si su *competidor* libre se convierte en el estándar del mercado y es demandado por sus propios clientes.

Ejemplo

Son muchos los casos donde se usa este modelo de negocio, ya que es común considerarlo como menos arriesgado que el de conocimiento *puro*. Sin embargo, las empresas que lo han usado han tenido evoluciones variadas. Algunas de ellas son:

- Caldera

La historia de Caldera es complicada. En sus inicios, creó su propia distribución de GNU/Linux, orientada a las empresas: Caldera OpenLinux. En el 2001, compró la división de Unix de SCO, y en el 2002 cambió su nombre a "SCO Group". Su estrategia empresarial ha dado tantos vuelcos como su nombre, desde su total apoyo a Linux hasta sus demandas contra IBM y Red Hat en el 2003, y su abandono de su propia distribución. Pero en lo que se refiere a este apartado, el negocio de Caldera, al menos hasta el 2002, es un claro exponente del mejor conocimiento con limitaciones. Caldera trataba de explotar su conocimiento de la plataforma GNU/Linux, pero limitando la competencia a que podía verse sometida mediante la inclusión de software propietario en su distribución. Esto hacía difícil a sus clientes cambiar de distribución una vez que la habían adoptado, pues aunque las demás distribuciones

Lectura complementaria

[Ver bibliografía, apartado 15]

ANOTACIONES

de GNU/Linux incluían la parte libre de Caldera OpenLinux, no encontraban en ellas la parte propietaria.

- Ximian

Fundada en 1999 con el nombre de Helix Code por desarrolladores muy vinculados al proyecto GNOME, fue adquirida en agosto del 2003 por Novell. La mayor parte del software que ha desarrollado ha sido libre (en general, parte de GNOME). Sin embargo, en un ámbito muy concreto Ximian decidió licenciar un componente como software propietario: el Connector for Exchange. Éste es un módulo que permite a uno de sus productos estrella, Evolution (un gestor de información personal que incluye correo electrónico, agenda, calendario, etc.) interactuar con servidores MS Exchange, muy utilizados en grandes organizaciones. De esta manera, trata de competir ventajosamente con otras empresas que proporcionan servicios basados en GNOME, quizás con los productos desarrollados por la propia Ximian, que no podrán interaccionar tan fácilmente con Exchange. Salvo por este producto, el modelo de Ximian ha sido de *mejor conocimiento*, y también basado en ser la fuente de un programa (ver más adelante).

Lectura complementaria

[Ver bibliografía, apartado 58]

5.2.3. Fuente de un producto libre

Este modelo es similar al basado en el mejor conocimiento, pero especializándolo de forma que la empresa que lo utiliza es productora, casi en su totalidad, de un producto libre. Naturalmente, la ventaja competitiva aumenta al ser éstos los desarrolladores del producto en cuestión, controlar su evolución, y tenerlo antes que la competencia. Todo ello posiciona a la empresa desarrolladora en un lugar muy bueno de cara a los clientes que deseen servicios sobre ese programa. Además, es un modelo muy interesante en términos de imagen, ya que la empresa ha demostrado su potencial desarrollador con la creación y mantenimiento de la aplicación en cuestión, lo que puede ser muy interesante de cara

ANOTACIONES

a convencer a posibles clientes de sus capacidades. Igualmente, proporciona muy buena imagen de cara a la comunidad del software libre en general, ya que reciben de la empresa un nuevo producto libre que pasa a formar parte del acervo común.

Ejemplo

Son muchos los productos libres que comenzaron su desarrollo dentro de una empresa, y muy habitualmente ha continuado siendo esa empresa quien ha guiado su desarrollo posterior. A modo de ejemplo, podemos citar los siguientes casos:

- Ximian

Ya hemos mencionado cómo, en parte, ha usado el modelo de mejor conocimiento con limitaciones. Pero, en general, Ximian ha seguido un claro modelo basado en ser la fuente de programas libres. Sus principales productos, como Evolution, RedCarpet o Mono se han distribuido bajo licencias de GNU, y Ximian los ha desarrollado casi en exclusiva desde el comienzo. La empresa ha tratado de rentabilizar este desarrollo consiguiendo contratos para hacerlos evolucionar en ciertos sentidos, para adaptarlos a las necesidades de sus clientes y ofreciendo personalización y mantenimiento.

- Zope Corporation

En 1995 se funda Digital Creations, que desarrolla un producto propietario para la gestión de anuncios clasificados vía web. En 1997 recibió una inyección de capital por parte, entre otros, de una empresa de capital-riesgo, Opticality Ventures. Lo extraño (en aquella época) de esta inversión es que le pusieron como condición que distribuyera como software libre la evolución de su producto, lo que más adelante fue Zope, uno de los gestores de contenidos más populares en Internet. El modelo de negocio de la empresa desde entonces fue producir Zope y productos relacionados con él, y ofrecer servicios de adaptación y mantenimiento para todos ellos. Zope Corporation

Lectura complementaria

[Ver bibliografía, apartado 59]

ANOTACIONES

ha sabido, además, crear una dinámica comunidad de desarrolladores de software libre alrededor de sus productos, y colaborar activamente con ellos.

5.2.4. Fuente de un producto con limitaciones

Este modelo es similar al anterior, pero incluye medidas para limitar la competencia o maximizar los ingresos. Entre las limitaciones más habituales podemos considerar las siguientes:

- Distribución propietaria durante un tiempo, luego libre

Con o sin promesa de distribución libre posterior, cada nueva versión del producto se vende como software propietario. Pasado un tiempo (normalmente, cuando se empieza a comercializar una nueva versión, también como software propietario), esa versión pasa a distribuirse con una licencia libre. De esta manera, la empresa productora obtiene ingresos de los clientes interesados en disponer lo antes posible de nuevas versiones, y a la vez minimiza la competencia, ya que cualquier empresa que quiera competir usando ese producto sólo podrá hacerlo con la versión libre (disponible sólo cuando ya existe una nueva versión propietaria, supuestamente mejor y más completa).

- Distribución limitada durante un tiempo

En este caso, el software es libre desde que se comienza a distribuir. Pero como nada en una licencia libre obliga a distribuir el programa a quien lo quiera (esto es algo que quien tiene el software puede hacer o no), lo que hace el productor es distribuirlo durante un tiempo sólo a sus clientes, que le pagan por ello (normalmente en forma de contrato de mantenimiento). Al cabo de un tiempo, el productor lo distribuye a cualquiera, por ejemplo poniéndolo en un archivo de acceso público. De esta manera, el productor obtiene ingresos de sus clientes, que perciben esta disposición preferente del software como un valor añadido. Naturalmente, el modelo sólo funciona si los clientes a su vez no hacen público el programa cuando lo reciben. Para cierto tipo de clientes, esto puede no ser habitual.

ANOTACIONES

En general, en estos casos las empresas desarrolladoras obtienen los beneficios mencionados, pero no a coste cero. Debido al retraso con el que el producto está disponible para la comunidad del software libre, es prácticamente imposible que ésta pueda colaborar en su desarrollo, por lo que el productor se beneficiará muy poco de contribuciones externas.

Ejemplo

Algunos casos de empresas que utilizan este modelo de negocio son lo siguientes:

- artofcode LLC

Desde el año 2000, artofcode comercializa Ghostscript en tres versiones (anteriormente lo hacía Alladin Enterprises, con un modelo similar). La versión más actual la distribuye como AFPL Ghostscript, bajo una licencia propietaria (que permite el uso y la distribución no comercial). La siguiente (con un retraso de un año, más o menos) la distribuye como GNU Ghostscript, bajo la GNU GPL. Por ejemplo, en el verano del 2003, la versión AFPL es la 8.11 (liberada el 16 de agosto), mientras que la versión GNU es la 7.07 (distribuida como tal el 17 de mayo, pero cuya versión AFPL equivalente es del 2002). Además, artofcode ofrece una tercera versión, con una licencia propietaria que permite la integración en productos no compatibles con la GNU GPL (en este caso usa un modelo dual, que será descrito más adelante).

- Ada Core Technologies

Fue fundada en 1994 por los autores del primer compilador de Ada 95, cuyo desarrollo fue financiado en parte por el Gobierno de EE.UU., y que estaba basado en GCC, el compilador de GNU. Desde el principio sus productos han sido software libre. Pero la mayoría de ellos los ofrecen primero a sus clientes, como parte de un contrato de mantenimiento. Por ejemplo, su compilador, que sigue estando basado en GCC y se distribu-

Lectura complementaria

[Ver bibliografía, apartado 9]

Lectura complementaria

[Ver bibliografía, apartado 2]

ANOTACIONES

ye bajo la GNU GPL, se ofrece a sus clientes como Gnat Pro. Ada Core Technologies no ofrece este compilador al público en general de ninguna manera, y normalmente no se encuentran versiones del mismo en la Red. Sin embargo, con un retraso variable (en torno a un año), Ada Core Technologies ofrece las versiones *públicas* de su compilador, muy similares pero sin ningún tipo de soporte, en un archivo de ftp anónimo.

5.2.5. Licencias especiales

En estos modelos, la empresa produce un producto que distribuye bajo dos licencias o más. Al menos una de ellas es de software libre, pero las otras son típicamente propietarias y le permiten vender el producto de una forma más o menos tradicional. Normalmente, estas ventas se complementan con la oferta de consultoría y desarrollos relacionados con el producto. Por ejemplo, una empresa puede distribuir un producto como software libre bajo la GNU GPL, pero ofrecer también una versión propietaria (simultáneamente, y sin retraso para ninguna de las dos) para quien no quiera las condiciones de la GPL, por ejemplo, porque quiere integrar el producto con uno propietario (algo que la GPL no permite).

Ejemplo

Sleepycat Software fue fundada en 1996, y anuncia que desde ese momento ha tenido beneficios (lo que desde luego es notable en una empresa relacionada con el software). Sus productos, incluyendo Berkeley DB (un gestor de datos muy popular y que puede empotrarse fácilmente en otras aplicaciones) se distribuyen bajo una licencia libre que especifica que en caso de empotrarse en otro producto, ha de ofrecerse el código fuente de ambos. Sleepycat ofrece servicios de consultoría y desarrollo para sus productos, pero además los ofrece bajo licencias que permiten empotrarlos sin tener que distribuir el código fuente. Naturalmente, esto lo hace bajo contrato específico, y en general en régimen de venta como software propietario.

Lectura complementaria

[Ver bibliografía, apartado 50]

ANOTACIONES

5.2.6. Venta de marca

Aunque puedan conseguirse productos muy similares por menos dinero, muchos clientes están dispuestos a pagar el extra por comprar una *marca*. Este principio es utilizado por empresas que invierten en establecer una marca con buena imagen, y bien reconocida, que más adelante les permita vender con suficiente margen productos libres. En muchos casos no sólo venden esos productos, sino que los acompañan de servicios que los clientes aceptarán también como valor añadido.

Los casos más conocidos de este modelo de negocio son las empresas que comercializan distribuciones GNU/Linux. Estas empresas tratan de vender algo que en general se puede obtener a un coste bastante menor en la Red (o en otras fuentes con menos imagen de marca). Por ello han de conseguir que el consumidor reconozca su marca, y esté dispuesto a pagar el sobreprecio. Para ello, no sólo invierten en publicidad, sino que también ofrecen ventajas objetivas (por ejemplo, una distribución bien conjuntada o un canal de distribución que llegue hasta las cercanías del cliente). Además, suelen ofrecer a su alrededor una gran cantidad de servicios, tratando de rentabilizar el máximo posible esa imagen de marca (desde formación hasta programas de certificación para terceras partes).

Ejemplo

La distribución Red Hat Linux comenzó a distribuirse en 1994 (la empresa empezó a conocerse con el nombre actual en 1995). Desde entonces, Red Hat ha conseguido posicionar su nombre como el de *la* distribución de GNU/Linux por excelencia. Hoy día, en torno a ese nombre Red Hat comercializa todo tipo de servicios relacionados con la distribución, con GNU/Linux, y con software libre en general.

Lectura complementaria

[Ver bibliografía, apartado 46]

5.3. Otras clasificaciones de modelos de negocio

Hay otras clasificaciones de modelos de negocio clásicas en la literatura sobre software libre. A modo de ejemplo ofrecemos a continuación una de las más clásicas.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 120]

5.3.1. Clasificación de Hecker

La clasificación que se ofrece en Frank Hecker (mayo, 1998). *Setting up shop: The business of open-source software* fue la más usada por la publicidad de la Open Source Initiative, y también una de las primeras en tratar de categorizar los negocios que estaban surgiendo por aquella época. Sin embargo, incluye varios modelos poco centrados en software libre (en ellos es poco más que un acompañante del modelo principal). En cualquier caso, los modelos que describe son los siguientes:

- *Support seller* (venta de servicios relacionados con el producto).

La empresa promueve un producto libre (que ha desarrollado o en cuyo desarrollo participa activamente) y vende servicios, como consultoría o adaptación a necesidades concretas para él.

- *Loss leader* (venta de otros productos propietarios).

En este caso, el programa libre se utiliza para promover de alguna forma la venta de otros productos propietarios relacionados con él.

- *Widget frosting* (venta de hardware).

El negocio fundamental es la venta de hardware, y el software libre se considera un complemento para él, que puede ayudar a la empresa a obtener una ventaja competitiva.

- *Accessorizing* (venta de accesorios).

Se comercializan productos relacionados con el software libre, como libros, dispositivos informáticos, etc.

- *Service enabler* (venta de servicios).

El software libre sirve para crear un servicio (normalmente accesible en línea) del que la empresa obtiene algún beneficio.

ANOTACIONES

- *Brand licensing* (venta de marca).

Una empresa registra marcas que consigue asociar con programas libres, probablemente desarrollados por ella. Luego obtiene ingresos cuando vende derechos del uso de esas marcas.

- *Sell it, free it* (vende, libera).

Modelo similar a *loss leader*, pero realizado de forma cíclica. Primero se comercializa un producto como software libre. Si se consigue que tenga cierto éxito, la siguiente versión se distribuye como software propietario durante un tiempo, y pasado éste también se libera. Para entonces, se empieza a distribuir una nueva versión propietaria, y así sucesivamente.

- *Software franchising* (franquicia de software).

Una empresa franquicia el uso de sus marcas, relacionadas con un programa libre determinado.

Nota

El lector habrá podido observar cómo esta clasificación es bastante diferente de la que hemos ofrecido nosotros, pero aun así algunas de sus categorías coinciden casi exactamente con algunas de las nuestras.

5.4. Impacto sobre las situaciones de monopolio

El mercado informático tiende a la dominación de un producto en cada uno de sus segmentos. Los usuarios quieren rentabilizar el esfuerzo realizado en aprender cómo funciona un programa, las empresas quieren encontrar gente formada en el uso de su software, y todos quieren que los datos que gestionan puedan ser entendidos por los programas de las empresas y por las personas con las que se relacionan. Por esta razón, cualquier iniciativa dedicada a romper una situación *de facto* donde un producto domina claramente el mercado está destinada a producir más de lo mismo: si tiene éxito,

ANOTACIONES

vendrá otro a ocupar ese hueco, y en breve tendremos un nuevo dominante. Sólo los cambios tecnológicos producen, durante un tiempo, la inestabilidad suficiente como para que nadie domine claramente.

Pero el que haya un producto dominante no ha de llevar necesariamente a la constitución de un monopolio empresarial. Por ejemplo, la gasolina es un producto que casi domina el mercado de combustibles para turismos, pero (en un mercado de la gasolina libre) hay muchas empresas que producen y distribuyen este producto. En realidad, cuando hablamos de software, lo preocupante es lo que ocurre cuando un producto llega a dominar el mercado, porque ese producto tiene una sola empresa proveedora posible. El software libre ofrece una alternativa a esta situación: los productos libres pueden estar promovidos por una empresa en concreto, pero esta empresa no los controla, o al menos no lo hace hasta el extremo al que nos tiene acostumbrados el software propietario. En el mundo del software libre, un producto dominante no supone necesariamente un monopolio de empresa. Por el contrario, sea el que sea el producto que domina el mercado, muchas empresas pueden competir en proporcionarlo, mejorarlo, adaptarlo a las necesidades de sus clientes y ofrecer servicios alrededor de él.

5.4.1. Elementos que favorecen los productos dominantes

En informática es muy común que haya un producto claramente dominante en cada segmento de mercado. Y eso es normal por varios motivos, entre los que cabe destacar los siguientes:

- Formatos de datos

En muchos casos el formato de datos está fuertemente ligado a una aplicación. Cuando un número suficientemente alto de gente la usa, su formato de datos se convierte en estándar *de facto*, y las presiones para usarlo (y la aplicación por tanto) son formidables.

- Cadenas de distribución

Normalmente, uno de los problemas para empezar a usar un programa es obtener una copia de él. Y normalmente es difícil encontrar

ANOTACIONES

programas que no son líderes en su mercado. Las cadenas de distribución son costosas de mantener, de forma que los competidores minoritarios lo tienen difícil para llegar a la tienda de informática, donde el usuario final pueda comprarlos. El producto dominante, sin embargo, lo tiene fácil: el primer interesado en tenerlo va a ser la propia tienda de informática.

- Marketing

El marketing "gratuito" que obtiene un producto una vez que lo usa una fracción significativa de una población determinada es enorme. El boca a oreja funciona muy bien, también preguntar e intercambiar información con los conocidos. Pero, sobre todo, el impacto en los medios es muy grande: las revistas de informática hablarán una y otra vez de un producto si parece ser el que más se usa. Habrá cursos de formación para él, libros que lo describan, entrevistas a sus usuarios, etc.

- Inversión en formación

Una vez se han invertido tiempo y recursos en aprender cómo funciona una herramienta, se está muy motivado para no cambiar. Además, usualmente esa herramienta es la que ya domina el mercado, porque es más fácil encontrar personal y material que ayuden a aprender a usarla.

- Software preinstalado

Recibir una máquina con software ya instalado desde luego es un gran incentivo para usarlo, incluso si hay que pagar por él aparte. Y normalmente, el tipo de software que el vendedor de la máquina va a estar dispuesto a preinstalar será solamente el más utilizado.

5.4.2. El mundo del software propietario

En el mundo del software propietario la aparición de un producto dominante en un segmento cualquiera equivale a un monopolio por parte de la empresa que lo produce. Por ejemplo, tenemos estas situaciones monopolísticas *de facto* (o casi) de producto y empresa en

ANOTACIONES

los mercados de sistemas operativos, autoedición, bases de datos, diseño gráfico, procesadores de textos, hojas de cálculo, etc.

Y esto es así porque la empresa en cuestión tiene un gran control sobre el producto líder. Tan grande, que sólo ellos pueden marcar la evolución del producto, las líneas fundamentales donde se va a desarrollar, su calidad, etc. Los usuarios tienen muy poco control, dado que estarán muy poco motivados para probar con otros productos (por los motivos que se han comentado en el apartado anterior). Ante esto, poco podrán hacer, salvo tratar de desafiar la posición dominante del producto mejorando excepcionalmente los suyos (para tratar de contrarrestar esos mismos motivos), normalmente con poco éxito.

Esta situación pone a todo el sector en manos de la estrategia de la empresa dominante. Todos los actores dependen de ella, e incluso el desarrollo de la tecnología software en ese campo estará mediatizado por las mejoras que le haga a su producto. En el caso general, ésta es una situación donde aparecen los peores efectos económicos del monopolio, y en particular, la falta de motivación de la empresa líder para acercar el producto a las necesidades (siempre en evolución) de sus clientes. Éstos se han convertido en un mercado cautivo.

5.4.3. La situación con software libre

Sin embargo, en el caso del software libre un producto dominante no se traduce automáticamente en un monopolio de empresa. Si el producto es libre, cualquier empresa puede trabajar con él, mejorarlo, adaptarlo a las necesidades de un cliente y en general, ayudar en su evolución. Además, precisamente por su posición dominante, serán muchas las empresas interesadas en trabajar con él. Si el productor "original" (la empresa que desarrolló originalmente el producto) quiere permanecer en el negocio ha de competir con todas ellas, y por eso estará muy motivado para hacer evolucionar el producto precisamente en la línea que sus usuarios quieran. Naturalmente, tendrán la ventaja de un mejor conocimiento del programa, pero eso es todo. Tienen que competir por cada cliente.

La aparición de productos dominantes se traduce en el mundo del software libre, por lo tanto, en mayor competencia entre em-

ANOTACIONES

presas. Y con ello los usuarios retoman el control: las empresas en competencia no pueden más que hacerles caso si quieren sobrevivir. Y precisamente esto es lo que asegurará que el producto mejore.



Productos libres que son dominantes en su sector

Apache es desde hace tiempo líder en el mercado de servidores de web. Pero hay muchas empresas que están detrás de Apache, desde algunas muy grandes (como IBM) a otras muy pequeñas. Y todas ellas no tienen más remedio que competir mejorándolo, y normalmente contribuyendo al proyecto con sus mejoras. A pesar de que Apache es casi un monopolio en muchos ámbitos (por ejemplo, es casi el único servidor web que se considera sobre la plataforma GNU/Linux o *BSD), no depende de una sola empresa, sino de literalmente decenas de ellas.

Las distribuciones de GNU/Linux son también un caso interesante. GNU/Linux no es desde luego un monopolio, pero es posiblemente la segunda opción en el mercado de sistemas operativos. Y ello no ha forzado la situación donde una empresa tenga su control. Al contrario, hay decenas de distribuciones, realizadas por empresas diferentes, que compiten libremente en el mercado. Cada una de ellas trata de ofrecer mejoras que sus competidores tienen que adoptar a riesgo de ser echados del mercado. Pero además no pueden separarse demasiado de lo que es “GNU/Linux estándar”, pues eso es rechazado por los usuarios como una “salida de la norma”. La situación después de varios años de crecimiento de la cuota de mercado de GNU/Linux nos muestra a decenas de empresas compitiendo y haciendo evolucionar el sistema. Y de nuevo todas ellas están detrás de satisfacer las necesidades de sus usuarios. Sólo así pueden mantenerse en el mercado.

ANOTACIONES

GCC es un producto dominante en el mundo de compiladores C y C++ para el mercado GNU/Linux. Y sin embargo, eso no ha llevado a ninguna situación de monopolio de empresa, incluso cuando Cygnus (hoy RedHat) se ha encargado durante mucho tiempo de coordinar su desarrollo. Hay muchas empresas que hacen mejoras al sistema, y todas ellas compiten, cada una en su nicho específico, por satisfacer las demandas de sus usuarios. De hecho, cuando alguna empresa u organización específica ha fallado en el trabajo de coordinación (o así lo ha percibido una parte de los usuarios) ha habido espacio para un *fork* (división) del proyecto, con dos productos en paralelo durante un tiempo, hasta que eventualmente han vuelto a unirse (como está ocurriendo ahora con GCC 3.x).

5.4.4. Estrategias para constituirse en monopolio con software libre

A pesar de que el mundo del software libre es mucho más hostil a los monopolios de empresa que el mundo del software propietario, hay estrategias que una empresa puede utilizar para tratar de aproximarse a una situación de dominación monopolística de un mercado. Éstas son prácticas comunes en muchos otros sectores económicos, y para evitarlas trabajan las entidades de regulación de la competencia, por lo que no hablaremos de ellas en detalle. Sin embargo, sí que mencionaremos una que es hasta cierto punto específica del mercado del software, y que ya está siendo experimentada en algunas situaciones: la aceptación de productos certificados por terceros.

Cuando una empresa quiere distribuir un producto software (libre o propietario) que funcione en combinación con otros, es común "certificar" ese producto para una cierta combinación. El fabricante se compromete a ofrecer servicios (actualización, soporte, resolución de problemas, etc.) sólo si el cliente le asegura que está usando el producto en un entorno certificado por él. Por ejemplo, un fabricante de gestores de bases de datos puede certificar su producto para una determinada distribución de Linux, y para ninguna otra. Eso implicará

ANOTACIONES

que sus clientes tendrán que usar esa distribución de Linux u olvidarse del soporte por parte del fabricante (lo que, si el producto es propietario, puede ser imposible en la práctica). Si un fabricante dado consigue una posición claramente dominante como producto certificado de terceras partes, los usuarios no van a tener otras posibilidades más que la de utilizar ese producto. Si en ese segmento la certificación es importante, estamos de nuevo ante una situación de empresa monopolística.

Nota

Hasta cierto punto, en el mercado de distribuciones de GNU/Linux se están empezando a observar ciertos casos de situaciones tendentes al monopolio *de facto* en la certificación. Por ejemplo, hay muchos fabricantes de productos propietarios que sólo certifican sus productos sobre una distribución de GNU/Linux dada (muy habitualmente Red Hat Linux). Por el momento esto no parece redundar en una situación monopolística por parte de ninguna empresa, lo que podría ser debido a que en el mercado de distribuciones de GNU/Linux la certificación tenga poca importancia para los usuarios. Pero sólo el futuro dirá si en algún momento esta situación se acerca a una de monopolio *de facto*.

Sin embargo, es importante tener en cuenta dos comentarios con respecto a lo dicho. El primero es que estas posiciones monopolísticas no serán fáciles de conseguir, y en cualquier caso lo serán por mecanismos en general "no informáticos" (a diferencia de la situación de producto dominante, que como ya vimos es relativamente normal y se llega a ella por medio de mecanismos puramente relacionados con la informática y sus patrones de uso). El segundo es que si todo el software utilizado es libre, esta estrategia tiene pocas posibilidades de éxito (si tiene alguna). Un fabricante podrá conseguir que muchas empresas certifiquen sus productos, pero los clientes siempre podrán buscar fuentes de servicio y soporte diferentes de esas empresas que han certificado para él, si así lo consideran convenientes.

ANOTACIONES

6. Iniciativas públicas

Las instituciones públicas, tanto las que tienen capacidad legislativa como las que se dedican a administrar el Estado (las “administraciones públicas”) tienen en general mucho que decir en lo que se refiere a adopción y promoción de tecnologías. Aunque desde el comienzo de su historia hasta prácticamente el año 2000 no había habido (salvo casos casi anecdóticos) interés por parte de estas instituciones por el fenómeno del software libre, la situación está cambiando radicalmente a partir de esa fecha. En primer lugar, las administraciones públicas cada vez son más usuarias del software libre. En segundo, en su papel de promotoras de la sociedad de la información, cada vez son más las administraciones que están promoviendo directa o indirectamente el desarrollo de software libre. En tercer lugar, los entes legislativos se van poco a poco ocupando del software libre, aunque no siempre para mejorar el entorno donde se desarrolla.

Dado que el mundo del software libre se desarrolló durante tanto tiempo al margen del apoyo explícito (e incluso del interés) de las instituciones públicas, la situación actual causa sus polémicas y sus problemas, no sólo entre otros actores del sector de las tecnologías de la información, sino también dentro de la comunidad del software libre. En este capítulo trataremos de exponer cuál es la situación actual, y qué peculiaridades tiene el software libre cuando se relaciona con “lo público”.

6.1. Impacto del software libre en las administraciones públicas

El impacto potencial del software libre en las administraciones públicas, tanto en su faceta de consumidores como en su faceta de promotores (que se estudiarán con más detalle más adelante, apartado 6.2) presenta ciertos aspectos generales, que vamos a estudiar en este apartado.

ANOTACIONES

6.1.1. Impactos principales

Los impactos principales del software libre, y las principales nuevas perspectivas que permite, son los siguientes:

- Aprovechamiento más adecuado de los recursos

Muchas aplicaciones utilizadas o promovidas por las administraciones públicas son también utilizadas por muchos otros sectores de la sociedad. Por ello, cualquier inversión pública en el desarrollo de un producto libre que le interesa redundará en beneficios no sólo en la propia administración, sino en todos los ciudadanos que podrán usar ese producto para sus tareas informáticas, probablemente con las mejoras aportadas por la administración.

Nota

Un caso muy especial, pero de gran impacto, que muestra claramente este aprovechamiento de los recursos públicos puede encontrarse en la localización (adaptación a los usos y costumbres de una comunidad) de un programa. Aunque el aspecto más visible de la localización es la traducción de todos los textos que utiliza el programa, y de su documentación, hay otros muchos que son también afectados por ella (desde el uso de símbolos para la moneda local o la presentación de la fecha y hora en formatos habituales en la comunidad en cuestión hasta el uso de ejemplos y formas de expresión adecuadas a las costumbres locales en la documentación).

En cualquier caso, es claro que si una administración pública dedica recursos a que una determinada aplicación sea localizada para sus necesidades, es más que probable que esas necesidades coincidan con las de sus ciudadanos. De forma que no sólo consigue tener un programa informático que satisfaga sus necesidades, sino que también puede, sin gasto extra, ponerlo a disposición de cualquier ciudadano que pueda utilizarlo con provecho. Por ejemplo, cuando una administración pública financia la adaptación de

ANOTACIONES

un programa ofimático a una lengua que se habla en su ámbito de actuación, no sólo podrá usarlo en sus propias dependencias, sino que también podrá ofrecerlo a sus ciudadanos, con lo que ello puede suponer de fomento de la sociedad de la información.

- Fomento de la industria local

Una de las mayores ventajas del software libre es la posibilidad de desarrollar industria local de software. Cuando se usa software propietario, todo lo gastado en licencias va directamente al fabricante del producto, y además esa compra redunda en el fortalecimiento de su posición. Lo cual no es necesariamente perjudicial, pero poco eficiente para la región vinculada a la administración si analizamos la alternativa de usar un programa libre.

En este caso, las empresas locales podrán competir proporcionando servicios (y el propio programa) a la administración, en condiciones muy similares a cualquier otra empresa. Digamos que, de alguna manera, la administración está allanando el campo de juego, y haciendo más fácil que cualquiera pueda competir en él. Y naturalmente, entre esos *cualquiera* estarán las empresas locales, que tendrán la posibilidad de aprovechar sus ventajas competitivas (mejor conocimiento de las necesidades del cliente, cercanía geográfica, etc.).

- Independencia de proveedor

Es obvio que cualquier organización preferirá depender de un mercado en régimen de competencia que de un solo proveedor que puede imponer las condiciones en que proporciona su producto. Sin embargo, en el mundo de la administración esta preferencia se convierte en requisito fundamental, y hasta en obligación legal en algunos casos. La administración no puede, en general, elegir contratar con un suministrador dado, sino que debe especificar sus necesidades de forma que cualquier empresa interesada, que cumpla unas ciertas características técnicas, y que proporcione el servicio o el producto demandado con una cierta calidad, pueda optar a un contrato.

ANOTACIONES

De nuevo, en el caso del software propietario, para cada producto no hay más que un proveedor (aunque use una variedad de intermediarios). Si se especifica un producto dado, se está decidiendo también qué proveedor contratará con la administración. Y en muchos casos es prácticamente imposible evitar especificar un cierto producto cuando estamos hablando de programas de ordenador. Razones de compatibilidad dentro de la organización, o de ahorros en formación y administración, u otros muchos, hacen habitual que una administración decida usar un cierto producto.

La única salida a esta situación es que el producto especificado sea libre. En ese caso, cualquier empresa interesada podrá proporcionarlo, y también cualquier tipo de servicio sobre él (sujeto únicamente a sus capacidades y conocimientos del producto). Además, en caso de contratar de esta manera, la administración pertinente podrá en el futuro cambiar inmediatamente de proveedor si así lo desea, y sin ningún problema técnico, pues aunque cambie de empresa, el producto que usará será el mismo.

- Adaptación a las necesidades exactas

Aunque la adaptación a sus necesidades exactas es algo que necesita cualquier organización que precisa de la informática, las peculiaridades de la administración hacen que éste sea un factor muy importante para el éxito de la implantación de un sistema informático. En el caso de usar software libre, la adaptación puede hacerse con mucha mayor facilidad, y lo que es más importante, sirviéndose de un mercado con competencia, si hace falta contratarla.

Cuando una administración compra un producto propietario, modificarlo pasa normalmente por alcanzar un acuerdo con su productor, que es el único que legalmente (y muchas veces técnicamente) puede hacerlo. En esas condiciones, es difícil realizar buenas negociaciones, sobre todo si el productor no está excesivamente interesado en el mercado que le ofrece la administración en cuestión. Sin embargo, usando un producto libre, esa administración puede modificarlo a su antojo, si dispone de personal para ello, o contratar externamente la modificación. Esta contratación la puede realizar en principio cualquier empresa que tenga los conocimientos y capacidades para ello, por lo que es de esperar que haya concurrencia de varias empresas.

ANOTACIONES

Este hecho, necesariamente tiende a abaratar los costes y a mejorar la calidad.

- Escrutinio público de seguridad

Para una administración pública poder garantizar que sus sistemas informáticos hacen sólo lo que está previsto que hagan es un requisito fundamental, y en muchos estados, un requisito legal. No son pocas las veces que esos sistemas manejan datos privados, en los que pueden estar interesados muchos terceros (pensemos en datos fiscales, penales, censales, electorales, etc.). Difícilmente, si se usa una aplicación propietaria sin código fuente disponible, puede asegurarse que efectivamente esa aplicación trata esos datos como debe. Pero incluso si se ofrece su código fuente, las posibilidades que tendrá una institución pública para asegurar que no contiene código *extraño* serán muy limitadas. Sólo si se puede encargar ese trabajo de forma habitual y rutinaria a terceros, y cualquier parte interesada puede escrutarlos, podrá estar la administración razonablemente segura de cumplir con ese deber fundamental, o al menos de tomar todas las medidas en su mano para hacerlo.

- Disponibilidad a largo plazo

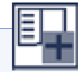
Muchos datos que manejan las administraciones y los programas que sirven para calcularlos han de estar disponibles dentro de decenas de años. Es muy difícil asegurar que un programa propietario cualquiera estará disponible cuando hayan pasado esos periodos de tiempo, y más si lo que se desea es que funcione en la plataforma habitual en ese momento futuro. Por el contrario, es muy posible que su productor haya perdido interés en el producto y no lo haya portado a nuevas plataformas, o que sólo esté dispuesto a hacerlo ante grandes contraprestaciones económicas. De nuevo, hay que recordar que sólo él puede hacer ese porte, y por lo tanto será difícil negociar con él. En el caso del software libre, por el contrario, la aplicación está disponible con seguridad para que cualquiera la porte y la deje funcionando según las necesidades de la administración. Si eso no sucede de forma espontánea, la administración siempre puede dirigirse a varias empresas buscando la mejor oferta para hacer el trabajo. De esta

ANOTACIONES

Lectura recomendada

[Ver bibliografía, apartado 192]

forma puede garantizarse que la aplicación y los datos que maneja estarán disponibles cuando haga falta.



Sugerencia

El lector interesado en un informe sobre las ventajas del software libre para la administración, escrito en el contexto estadounidense de 1999, puede consultar "The Case for Government Promotion of Open Source Software".

6.1.2. Dificultades de adopción

Si bien las ventajas de uso del software libre en las administraciones son muchas, también son muchas las dificultades cuando se enfrentan a su puesta en práctica. Entre éstas pueden destacarse:

- Desconocimiento y falta de decisión política

El primer problema que se encuentra el software libre para su introducción en las administraciones es uno que, sin duda, comparte con otras organizaciones: es aún muy desconocido entre quienes toman las decisiones.

Afortunadamente, éste es un problema que cada vez se va solucionando mejor, pero en muchos ámbitos de las administraciones el software libre aún es percibido como algo *extraño*, y tomar decisiones en la línea de usarlo implica asumir ciertos riesgos.

Unido a todo ello suele encontrarse un problema de decisión política. La principal ventaja del software libre en la administración no es el coste (siendo éste, de todas formas, importante, sobre todo cuando se habla de despliegues para gran cantidad de puestos). Las principales ventajas, como ya hemos visto, son sobre todo *de fondo*, estratégicas. Y por tanto quedan en gran medida en el ámbito de las decisiones políticas, no técnicas. Sin voluntad política de cambiar los sistemas informáticos y la filosofía con que

ANOTACIONES

se contratan, es difícil avanzar en la implantación de software libre en la administración.

- Poca adecuación de los mecanismos de contratación

Los mecanismos de contratación que se usan hoy día en la administración, desde los modelos de concurso público habituales hasta la división del gasto en partidas, están diseñados fundamentalmente para la compra de productos informáticos, y no tanto para la adquisición de servicios relacionados con los programas. Sin embargo, cuando se utiliza software libre, habitualmente no hay producto que comprar, o su precio es casi despreciable. Por el contrario, para aprovechar las posibilidades que ofrece el software libre, es conveniente poder contratar servicios a su alrededor. Esto hace preciso que, antes de utilizar seriamente software libre, se hayan diseñado mecanismos burocráticos adecuados que faciliten la contratación en estos casos.

- Falta de estrategia de implantación

En muchos casos, el software libre comienza a usarse en una administración simplemente porque el coste de adquisición es más bajo. Es muy habitual en estos casos que el producto en cuestión se incorpore al sistema informático sin mayor planificación, y en general sin una estrategia global de uso y aprovechamiento de software libre. Esto causa que la mayor parte de las ventajas del software libre se pierdan en el camino, ya que todo queda en el uso de un *producto más barato*, cuando ya hemos visto que en general las mayores ventajas son de otro tipo.

Si a esto unimos que el uso de software libre, si no se diseña el cambio a él adecuadamente, puede suponer unos costes de transición no despreciables, hace que experiencias aisladas, y fuera de un marco claro, de uso de software libre en la administración puedan resultar fallidas y frustrantes.

- Escasez o ausencia de productos libres en ciertos segmentos

La implantación de software libre en cualquier organización puede chocar con la falta de alternativas libres de la calidad adecuada para

ANOTACIONES

cierto tipo de aplicaciones. En esos casos, la solución es complicada: lo único que se puede hacer es tratar de promover la aparición del producto libre que se necesita.

Afortunadamente, las administraciones públicas están en una buena posición para estudiar seriamente si les conviene fomentar, o incluso financiar o cofinanciar el desarrollo de ese producto. Hay que recordar que entre sus fines suele estar, por ejemplo, que sus administrados puedan acceder mejor a la sociedad de la información, o el fomento del tejido industria local. Sin duda, la creación de muchos programas libres incidirá positivamente en ambos objetivos, por lo que al mero cálculo de coste/beneficio directo habría que sumar los beneficios indirectos que esta decisión causaría.

6.2. Actuaciones de las administraciones públicas en el mundo del software

Las administraciones públicas actúan sobre el mundo del software al menos de tres maneras:

- Comprando programas y servicios relacionados con ellas. Las administraciones, como grandes usuarios de informática, son un actor fundamental en el mercado del software.

- Promoviendo de diversas formas el uso (y la adquisición) de ciertos programas en la sociedad. Esta promoción se hace a veces ofreciendo incentivos económicos (desgravaciones fiscales, incentivos directos, etc.), a veces con información y recomendaciones, a veces por "efecto ejemplo".

- Financiando (directa o indirectamente) proyectos de investigación y desarrollo que están diseñando el futuro de la informática.

En cada uno de estos ámbitos el software libre puede presentar ventajas específicas (además de las ya descritas en el apartado anterior), interesantes tanto para la administración como para la sociedad en general.

ANOTACIONES

6.2.1. ¿Cómo satisfacer mejor las necesidades de las administraciones públicas?

Las administraciones públicas son grandes consumidoras de informática. En lo que al software se refiere, compran habitualmente tanto productos de consumo masivo (*off-the-shelf*), como sistemas a medida. Desde este punto de vista, son fundamentalmente grandes centros de compras, similares a las grandes empresas, aunque con sus propias peculiaridades. Por ejemplo, en muchos ámbitos se supone que las decisiones de compra de las administraciones públicas no han de tener en cuenta simplemente parámetros de coste frente a funcionalidad, sino que otros, como el impacto de la compra en el tejido industrial y social, o consideraciones estratégicas a largo plazo, pueden ser asimismo importantes.

En cualquier caso, lo habitual hoy día en lo que se refiere a software de consumo masivo es utilizar los productos propietarios líderes del mercado. La cantidad de recursos públicos que se gastan los ayuntamientos, las comunidades autónomas, la Administración central y las administraciones europeas en comprar licencias de Windows, Office u otros productos similares es ciertamente considerable. Pero cada vez más las soluciones libres están penetrando en este mercado y cada vez más se están considerando soluciones basadas en software libre para los servidores. Productos como OpenOffice, y GNU/Linux con GNOME o KDE son cada vez más considerados en el escritorio.

¿Qué se puede ganar con esta migración a software libre? Para ilustrarlo, consideremos el siguiente escenario. Supongamos que con una fracción de lo gastado en dos o tres productos propietarios “estrella” por todas las administraciones europeas (o probablemente las de cualquier estado desarrollado de tamaño medio), se podría promover un concurso público para que una empresa (o dos, o tres, o cuatro) mejorasen y adaptasen los programas libres ahora disponibles para que en el plazo de uno o dos años estuvieran listos para su uso masivo al menos para ciertas tareas típicas (si no lo estuvieran ya). Imaginad, por ejemplo, un esfuerzo coordinado, a escala nacional o europea, para que todas las administraciones participasen en un consorcio que se encargase de la gestión de estos concursos. En poco tiempo habría una industria “local” especializada en realizar estas mejoras y estas adaptaciones. Y las administraciones podrían elegir entre las tres o cuatro distribuciones libres producidas por esta

ANOTACIONES

industria. Para fomentar la competencia se podría recompensar económicamente a cada empresa según la cantidad de administraciones que eligiesen usar su distribución. Y todo el resultado de esta operación, al ser software libre, estaría también a disposición de empresas y usuarios individuales, que en muchos casos tendrían necesidades similares a las de las administraciones.

En el caso del software hecho a medida, el proceso habitual por el momento pasa normalmente por contratar con una empresa los programas necesarios bajo un modelo propietario. Todo el desarrollo realizado a petición de la administración es propiedad de la empresa que lo desarrolla. Y normalmente la administración contratante queda atada a su proveedor para todo lo que tenga que ver con mejoras, actualizaciones y soporte, en un círculo vicioso que dificulta mucho la competencia y ralentiza el proceso de modernización de las administraciones públicas. Lo que es peor, en muchos casos el mismo programa es vendido una y otra vez a administraciones similares, aplicando en cada nuevo caso los costes que habría supuesto volver a realizar el desarrollo desde cero.

Consideremos de nuevo un escenario que muestre cómo podrían ser las cosas de otra forma. Un consorcio de administraciones públicas con necesidades de un cierto software a medida podría exigir que el resultado obtenido fuera software libre. Ello permitiría que otras administraciones se beneficiasen también del trabajo, y a medio plazo estuvieran interesadas en colaborar con el consorcio para que se tuvieran en cuenta sus necesidades peculiares. Al ser el software resultante libre, no habría obligación de contratar las mejoras y adaptaciones al mismo proveedor, introduciendo de esta forma competencia en ese mercado (que hoy por hoy es casi cautivo). Y en cualquier caso, el coste final para cualquiera de las administraciones implicadas no sería nunca mayor que si se hubiera utilizado un modelo propietario.

¿Son estos escenarios ciencia ficción? Como se verá más adelante, ya hay tímidas iniciativas en direcciones similares a las que se exponen en ellos. Además de ayudar a crear y mantener una industria en el ámbito de la administración pública que realiza la compra, el software libre tiene más ventajas específicas en el entorno público. Por ejemplo, es la forma más efectiva de tener software desarrollado en lenguas minoritarias (preocupación esencial de muchas administra-

ciones públicas). También puede ayudar mucho en el mantenimiento de la independencia estratégica a largo plazo, y en asegurar el acceso a los datos que custodian las administraciones públicas al cabo de un cierto tiempo. Por todo ello, las entidades públicas están cada vez más interesadas en el software libre como usuarias.

Ejemplo

Algunos casos relacionados con administraciones alemanas

En julio del 2003 se liberó la primera versión estable de Kolab, un producto del proyecto Kroupware. Kolab es un sistema libre de ayuda informática al trabajo en grupo (*groupware*) basado en KDE. El motivo por el que mencionamos este proyecto en este punto es porque su origen fue un concurso del Bundesamt für Sicherheit in der Informationstechnik (BSI) del Gobierno alemán (podría traducirse como Agencia Federal de Seguridad en Tecnologías de la Información). El concurso pedía la provisión de una solución que interoperase con Windows y Outlook por un lado y GNU/Linux y KDE por otro. Entre las ofertas presentadas, ganó la propuesta conjunta de tres empresas, Erfrakon, Intevation y Klarälvdalens Datakonsult, que proponían una solución libre basada parcialmente en software ya desarrollado por el proyecto KDE, completado con desarrollos propios libres, que dieron lugar a Kolab.

En mayo del 2003 el Ayuntamiento de Munich (Alemania) aprobó la migración a GNU/Linux y aplicaciones de ofimática libres de todos los ordenadores usados como escritorios, unos 14.000. La decisión de hacerlo no fue sólo económica: también se tuvieron en cuenta aspectos estratégicos y cualitativos, según declararon sus responsables. En el exhaustivo estudio que se realizó previamente a la decisión, la solución finalmente elegida (GNU/Linux más OpenOffice, fundamentalmente) obtuvo 6.218 puntos (de un máximo de 10.000) frente a los poco más de 5.000 que obtuvo la solución "tradicional" basada en software de Microsoft.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 88]

En julio del 2003 se hizo público por parte del Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), del Ministerio alemán del Interior, el documento *Leitfaden für die Migration von Basissoftwarekomponenten auf Server- und Arbeitsplatzsystemen* ('Guía de migración para componentes de software básicos en servidores y estaciones de trabajo'), que ofrece un conjunto de orientaciones sobre cómo pueden migrar a soluciones basadas en software libre las entidades públicas alemanas. Estas orientaciones están diseñadas para que quien esté a cargo de la toma de decisiones pueda evaluar si conviene una migración a software libre, y en qué condiciones hacerla si toma esa decisión.

6.2.2. Promoción de software

Las entidades públicas dedican muchos recursos a la promoción de planes que incentivan el gasto en informática. Ésta es una herramienta formidable, que puede ayudar mucho a la extensión de nuevas tecnologías por la sociedad. Pero también es una herramienta peligrosa. Por ejemplo, puede no ser muy buena idea promover el uso de Internet en la sociedad recomendando un determinado navegador que fomente la posición de monopolio *de facto* de una empresa, lo que a la larga podría ser perjudicial para la sociedad que se está tratando de beneficiar.

De nuevo, el software libre puede ayudar en estas situaciones. En primer lugar, es neutro frente a fabricantes, ya que ninguno tiene la exclusiva de ningún programa libre. Si una administración quiere promover el uso de una familia de programas libres, puede realizar concursos, a los que se puede presentar cualquier empresa del sector, para gestionar su entrega a los ciudadanos, su mejora o ampliación con las funcionalidades que se desee, etc. En segundo lugar, puede ayudar mucho en los aspectos económicos. Por ejemplo, en muchos casos puede utilizarse la misma cantidad de recursos en adquirir una cierta cantidad de licencias de un programa propietario para su uso por los ciudadanos, o en adquirir

ANOTACIONES

una copia de uno libre, y contratar soporte o adaptaciones para él. O incluso en negociar con un productor de software propietario la compra de los derechos de su producto para convertirlo en software libre.

En otro ámbito, podemos imaginar que parte de la cantidad destinada a un programa de informatización de escuelas se dedica a crear una distribución de GNU/Linux adaptada a las necesidades de la docencia en enseñanza primaria. Y con el resto de los recursos, se contrata soporte para que el software sea mantenido en esas escuelas, de forma que no sea un simple "software florero", sino que realmente haya gente encargada de su correcto funcionamiento. De esta manera se cubren no sólo las necesidades del sistema educativo, sino que además se genera un mercado para empresas, habitualmente de ámbito local, capaces de ofrecer servicios de mantenimiento. Y por supuesto, se deja completamente abierto el camino al futuro: el software no quedará obsoleto en pocos años obligando a recomenzar desde cero, sino que se podrá ir actualizando incrementalmente, año tras año, manteniendo los beneficios del programa con una inversión similar.

Nota

El lector conocedor de las iniciativas públicas con respecto al software libre reconocerá en este ejemplo el caso de LinEx. A finales del 2001 la Junta de Extremadura (España) decidió utilizar una distribución de GNU/Linux para informatizar todos los colegios públicos de la región. Para ello, financió la construcción de LinEx, una distribución basada en Debian GNU/Linux que fue anunciada en primavera del 2002, y se encargó de que fuera un requisito en todos los concursos de adquisición de equipamiento informático para centros educativos. Además, inició programas de formación y capacitación de profesores, de creación de materiales docentes y de extensión de la experiencia a otros ámbitos. A mediados del 2003, la experiencia parece ser un éxito, que ya se está extendiendo institucionalmente a otras regiones (por ejemplo, a Andalucía, también en España, mediante el proyecto GuadaLinex).

ANOTACIONES

6.2.3. Fomento de la investigación

El software también tiene interesantes ventajas en política de I+D que merece la pena analizar. Con dinero público se están financiando los desarrollos de gran cantidad de software del que la sociedad no acaba beneficiándose ni siquiera indirectamente. Habitualmente, los programas públicos de fomento de la investigación y desarrollo financian total o parcialmente proyectos que crean programas sin atender a los derechos que va a tener el público sobre ellos. En muchos casos, los resultados, sin un plan adecuado de comercialización, simplemente quedan en algún cajón cubriéndose de polvo. En otros, las mismas personas que financiaron un programa vía impuestos acaban pagándolo de nuevo si lo quieren usar (ya que tienen que adquirir licencias de uso).

El software libre ofrece una opción interesante, que está siendo considerada con atención, poco a poco, por las autoridades encargadas de la política de innovación en muchas administraciones. Especialmente en los casos donde la investigación es precompetitiva (lo más habitual en los casos de financiación pública), que los programas resultantes sean libres permite que la industria en su conjunto (y por ende la sociedad) se beneficie grandemente del dinero público gastado en I+D en el campo del software. Donde una empresa puede ver un resultado de imposible comercialización, otra puede ver una oportunidad de negocio. Así, por un lado, se maximizan los resultados de los programas de investigación. Y por otro, se favorece la competencia entre las empresas que quieran utilizar los resultados de un proyecto, ya que todas ellas competirán a partir de los mismos programas resultado del proyecto.

Este modelo no es nuevo. En gran medida es el que ha permitido el desarrollo de Internet. Si las administraciones públicas exigen que los resultados de la investigación realizada con sus fondos sean distribuidos en forma de programas libres, no sería extraño que apareciesen, en diversas escalas, casos similares. O bien los resultados de esas investigaciones son malos o inútiles (y en ese caso debería replantearse la forma de selección de los proyectos), o bien la dinamización que supondría dejarlos listos para que cualquier empresa pueda convertirlos en producto permitiría desarrollos sencillamente impredecibles.

ANOTACIONES

6.3. Iniciativas legislativas

vEn los siguientes apartados se repasan algunas iniciativas que los poderes legislativos están llevando a cabo en diversos países del mundo, centrándonos en las iniciativas pioneras. Desde luego, esta lista de iniciativas no es exhaustiva. El lector interesado en completarla puede consultar GrULIC. *Legislación sobre el uso de software libre en el estado*, donde se citan gran cantidad de iniciativas por todo el mundo.

Lectura complementaria

[Ver bibliografía, apartado 112]

6.3.1. Proyectos de ley en Francia

En 1999 y 2000 se presentaron en Francia dos proyectos de ley relacionados con el software libre que fueron los pioneros de una larga serie de debates legislativos sobre la materia:

- vEl proyecto de ley 1999-495, propuesto por Laffitte, Trégouet y Cabanel, fue expuesto en el servidor web del Senado de la república francesa a partir de octubre de 1999. Tras un proceso de discusión pública a través de Internet que se prolongó durante dos meses, el proyecto sufrió algunas modificaciones. El resultado es el proyecto de ley número 2000-117, que abogaba por el uso obligatorio de software libre en la Administración, previendo excepciones y medidas transitorias para aquellos casos en los que no sea aún técnicamente posible, en un marco más general que intentaba generalizar en la Administración francesa el uso de Internet y del software libre.

Lectura complementaria

[Ver bibliografía, apartados 83 y 141]

- En abril de 2000 los diputados Jean-Yves Le Déaut, Christian Paul y Pierre Cohen propusieron una nueva ley, cuyo objetivo es similar al proyecto de Laffitte, Trégouet y Cabanel: reforzar las libertades y la seguridad del consumidor, así como mejorar la igualdad de derechos en la sociedad de la información.

Sin embargo, a diferencia del proyecto de ley de Laffitte, Trégouet y Cabanel, este otro no fuerza a utilizar software libre en la administración. Este proyecto de ley se centra en que el software utilizado en la Administración tenga el código fuente disponible, si bien no obliga a que éste se distribuya con licencias de software libre.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartado 92]

Para conseguir sus objetivos, los legisladores pretenden garantizar el "derecho a la compatibilidad" del software, proporcionando mecanismos que lleven a la práctica el principio de interoperabilidad plasmado en la Directiva del Consejo Europeo relativa a la protección jurídica de programas de ordenador.

Ninguno de los dos proyectos franceses se convirtió en ley, pero ambos han servido de inspiración a la mayoría de las iniciativas posteriores en todo el mundo, y por ello es especialmente interesante su estudio. El segundo (propuesto por Le Déaut, Paul y Cohen) perseguía la compatibilidad e interoperabilidad del software, haciendo hincapié en la disponibilidad del código fuente del software utilizado en la Administración. Sin embargo, no requiere que las aplicaciones desarrolladas sean software libre, entendido como aquel que se distribuye con licencias que garantizan la libertad de modificación, uso y redistribución del programa.

Más adelante (apartados 6.4.1 y 6.4.2) reproducimos de manera casi íntegra el articulado y la exposición de motivos de ambos proyectos de ley. En particular, las exposiciones de motivos son de especial interés, pues resaltan cuáles son los problemas que acechan hoy día a las administraciones públicas en relación con el uso de software en general.

6.3.2. Proyecto de ley en Brasil

Lectura complementaria

[Ver bibliografía, apartado 164]

El diputado Walter Pinheiro presentó en diciembre de 1999 un proyecto de ley sobre el software libre en la Cámara Federal de Brasil. Este proyecto (aún en tramitación en agosto del 2003) afecta a la utilización de software libre en la Administración pública y en las empresas privadas controladas accionarialmente por el Estado.

En él se recomienda el uso de software libre en estas entidades que no tenga restricciones en cuanto a su préstamo, modificación o distribución. El articulado de la ley describe pormenorizadamente qué se entiende por software libre y cómo deben ser las licencias que lo acompañen. Las definiciones coinciden con la definición clásica de software libre del proyecto GNU. En la exposición de motivos se repasa la historia del proyecto GNU, anali-

ANOTACIONES

zando sus ventajas y logros. Asimismo, se hace referencia a la situación actual del software libre, utilizando como ejemplo el sistema operativo GNU/Linux.

Una de las partes más interesantes, el artículo primero, deja bien claro el ámbito en el que se propone el uso de software libre (teniendo en cuenta que la definición que se ofrece en los artículos posteriores para "programa abierto" es, como ya se ha dicho, software libre):



"La Administración pública, en todos los niveles, los poderes de la República, las empresas estatales y de economía mixta, las empresas públicas y todos los demás organismos públicos o privados sujetos al control de la sociedad brasileña están obligados a utilizar preferentemente, en sus sistemas y equipamientos de informática, programas abiertos, libres de restricciones propietarias en cuanto a su cesión, modificación y distribución."

6.3.3. Proyectos de ley en Perú

Son varios los proyectos de ley relacionados con el uso del software libre en las administraciones públicas que se han propuesto en Perú. El primero y el más conocido fue propuesto por el congresista Edgar Villanueva Núñez en diciembre del 2001. En él se define software libre según la definición clásica de las cuatro libertades (dándole quizás una mayor precisión legal, con una definición que especifica seis características que ha de tener un programa libre), y propone su utilización exclusiva en la administración peruana:

Lectura complementaria

[Ver bibliografía, apartados 163 y 199]



"Artículo 2. Los poderes Ejecutivo, Legislativo y Judicial, los organismos descentralizados y las empresas donde el Estado posea mayoría accionaria, emplearán en sus sistemas y equipamientos de informática exclusivamente programas o software libres".

ANOTACIONES

Aunque un poco más adelante, los artículos 4 y 5 incluyen algunas excepciones a esta regla.

Lectura complementaria

[Ver bibliografía, apartado 127]

En su día esta propuesta de ley tuvo gran repercusión mundial. Por un lado, fue la primera vez en que se propuso el uso exclusivo del software libre en una administración. Pero incluso más importante para su repercusión que esa novedad fue el intercambio epistolar entre el congresista Villanueva y la representación de Microsoft en Perú, que realizó alegaciones a esta propuesta. También es interesante esta propuesta de ley por la posición que tomó al respecto la embajada de EE.UU., que incluso llegó a enviar por conducto oficial una notificación (adjuntando un informe elaborado por Microsoft) al Congreso peruano expresando su "preocupación sobre las recientes propuestas del Congreso de la República para restringir las compras por parte del Gobierno peruano de software de código abierto o software libre". Entre otros motivos, tanto las alegaciones de Microsoft como de la Embajada de EE.UU. trataban de mostrar cómo la propuesta de ley discriminaría a unas empresas frente a otras, y haría imposible las inversiones necesarias para la creación de una industria nacional de creación de software. Ante esto, Villanueva argumenta cómo su proposición de ley no discrimina ni favorece de ninguna manera a ninguna empresa en particular, pues no especifica quién podrá ser proveedor de la Administración, sino cómo (en qué condiciones) tendrá que realizarse la provisión de software. Esta argumentación es muy clara para entender cómo la promoción del software libre en la Administración no perjudica en ningún caso a la libre competencia entre las empresas suministradoras.

Lectura complementaria

[Ver bibliografía, apartado 200]

Más adelante, los congresistas peruanos Edgar Villanueva Núñez y Jacques Rodrich Ackerman presentaron un nuevo proyecto de ley, el número 2485 de 8 de abril del 2002, que en agosto del 2003 sigue su trámite parlamentario. Este proyecto es una evolución del proyecto de ley 1609, ya que recoge varios comentarios y mejoras sobre él, y puede considerarse como un buen ejemplo de proyecto de ley que propone el uso exclusivo de software libre en las administraciones públicas, salvo en ciertos casos excepcionales. Por su interés, incluimos su texto íntegro (apartado 6.4.3). En particular, su exposición de motivos es un buen resumen de las características que debería tener el software utilizado por las administraciones públicas, y cómo éstas las cumple mejor el software libre que el propietario.

ANOTACIONES

6.3.4. Proyectos de ley en España

En España ha habido varias iniciativas legislativas relacionadas con el software libre. Citamos a continuación algunas de ellas:

- **Decreto de medidas de impulso de la sociedad del conocimiento en Andalucía**

Una de las iniciativas legislativas más importantes (por haber entrado en vigor) que han tenido lugar en España ha sido sin duda la adoptada por Andalucía. En el decreto de medidas de impulso de la sociedad del conocimiento en Andalucía se trata el uso de software libre, fundamentalmente (pero no sólo) en el entorno educativo.

Lectura complementaria

[Ver bibliografía, apartado 80]

Entre otros detalles, fomenta el uso de software libre de forma preferente en los centros docentes públicos, obliga a que todo el equipamiento adquirido para estos centros sea compatible con sistemas operativos libres, y lo mismo para los centros de la Junta que ofrezcan acceso público a Internet.

- **Proposición de ley de software libre en el marco de la Administración pública de Cataluña**

En otras comunidades se han discutido proposiciones más ambiciosas, pero sin que hayan logrado la mayoría necesaria. Entre ellas, la más conocida es probablemente la debatida en el Parlamento de Cataluña, muy similar a la que el mismo partido (Esquerra Republicana de Catalunya) presentó en el Congreso de los Diputados, de la que se habla a continuación. Esta propuesta no prosperó cuando fue sometida a votación.

Lectura complementaria

[Ver bibliografía, apartado 198]

- **Proyecto de ley de Puigcercós Boixassa en el Congreso de los Diputados**

También ha habido una iniciativa en el Congreso de los Diputados, propuesta por Joan Puigcercós Boixassa (Esquerra Republicana de Catalunya). Esta iniciativa propone la preferencia del uso de software libre en la Administración del Estado, y en ese sentido es similar a otras iniciativas con ese fin. Sin embargo, tiene la peculiaridad interesante de hacer énfasis en la disposición de los programas libres lo-

Lectura complementaria

[Ver bibliografía, apartado 167]

ANOTACIONES

calizados para los idiomas cooficiales (en las comunidades autónomas que los tienen). La iniciativa no consiguió ser aprobada en el trámite parlamentario.

6.4. Textos de algunas propuestas legislativas y documentos relacionados

A continuación se ofrece el texto literal de algunas de las propuestas legislativas mencionadas en este capítulo, y de algunos documentos relacionados con éstas.

6.4.1. Proyecto de ley de Laffitte, Trégouet y Cabanel (Francia)

Lectura complementaria

[Ver bibliografía, apartado 140]

A continuación reproducimos una traducción al español del proyecto de ley realizado en octubre de 1999 por los senadores franceses Pierre Laffitte, René Trégouët y Guy Cabanel.

Exposición de motivos

(Se incluyen solamente aquellos párrafos relativos al software libre)

[...] Para garantizar la perennidad de los datos accesibles, facilitar su intercambio y asegurar el libre acceso de los ciudadanos a la información, es necesario que su utilización en la Administración no dependa de la buena voluntad de los fabricantes de software. Se necesitan sistemas libres cuya evolución pueda garantizarse gracias a la disponibilidad por parte de todos del código fuente utilizado por el fabricante.

El desarrollo del software libre es actualmente muy fuerte. Son muchas las grandes empresas de informática que reconocen que el futuro de su negocio no estará en vender software, sino en facilitar su uso mediante la prestación de servicios asociados.

Nuestro proyecto de ley prevé que después de un periodo transitorio definido por decreto, el uso de software libre en las administraciones públicas sea obligatorio.

ANOTACIONES

Sólo se podrá utilizar software propietario cuyo código fuente no esté disponible en casos concretos si media una autorización de una agencia del software libre [...].

Artículos

- **Artículo 1**. Sobre la inmaterialización de los intercambios de la información y datos entre las administraciones públicas.

Los servicios del Estado, las administraciones locales y los organismos públicos se asegurarán de intercambiar sus datos e informaciones en soporte electrónico y con redes electrónicas, a partir del primero de enero del 2002.

Las condiciones que regularán la situación transitoria entre el actual intercambio mediante papel y el intercambio sobre soporte y redes electrónicas se precisarán mediante decreto.

- **Artículo 2**. Sobre la desmaterialización de los procesos de los mercados públicos.

Con el fin de conseguir una gran transparencia y un acceso rápido a la información por parte de las empresas, las convocatorias de ofertas públicas así como los documentos anexos serán objeto de publicidad en soporte y redes electrónicas a partir del primero de enero del 2002. Asimismo, deberá responderse a las convocatorias de ofertas públicas en soporte y mediante redes electrónicas.

Un decreto determinará los mecanismos de transición a los procesos electrónicos.

- **Artículo 3**. Sobre las tecnologías abiertas.

Los servicios del Estado, las administraciones locales y los organismos públicos sólo podrán utilizar a partir del primero de enero del 2002, excepción hecha de lo dispuesto en el artículo 4, software cuyo uso y modificación sean libres y para los que el código fuente esté disponible.

Un decreto fijará las condiciones de transición.

ANOTACIONES

- **Artículo 4**. Sobre la agencia del software libre.

Se creará la Agencia del Software Libre. Estará encargada de informar a los servicios del Estado, a las administraciones locales y a los organismos públicos de las condiciones de aplicación de la presente ley. La Agencia determinará las licencias de utilización de software que se adecuan al marco fijado por la presente ley.

La Agencia velará por la interoperabilidad del software libre dentro de las administraciones públicas.

La Agencia realizará el inventario, por sectores de actividad, de las carencias del software cuyo uso y modificación sean libres y cuyo código fuente esté disponible. En función de este inventario, la Agencia autorizará a las administraciones públicas a no aplicar la presente ley.

La Agencia del software libre estará abierta a los internautas y sus decisiones deberán estar precedidas de consultas realizadas a través de Internet.

Se designará a una persona de enlace con la Agencia del software libre en cada prefectura.

Las modalidades de funcionamiento de la Agencia del software libre se establecerán mediante decreto.

- **Artículo 5**. Sobre la difusión de las modificaciones del software utilizado en el marco de la presente ley.

La Agencia del Software Libre velará, respetando los derechos de los autores, porque sean difundidas las modificaciones al software realizadas en el marco de la presente ley.

- **Artículo 6**.

Los gastos que realice el Estado con motivo de la presente ley serán compensados mediante un incremento de los derechos apuntados en los artículos 575 y 575A del código general de impuestos.

ANOTACIONES

6.4.2. Proyecto de ley de Le Déaut, Paul y Cohen (Francia)

A continuación mostramos la traducción al español del texto prácticamente íntegro del proyecto de ley presentado por Jean-Yves Le Déaut, Christian Paul y Pierre Cohe en abril del 2000:

Exposición de motivos

El incremento fulgurante de la utilización de las nuevas tecnologías de la información y de las telecomunicaciones precisa de un acompañamiento legislativo. Los servicios públicos y las administraciones locales deben convertirse en modelo y motor de la sociedad de la información que sea garante de las libertades individuales, de la seguridad del consumidor y de la igualdad de oportunidades en la materia que nos ocupa.

Varios ejemplos muestran que, a pesar de algunos progresos importantes realizados gracias a la acción del Gobierno en el área de la sociedad de la información, los servicios del Estado suelen utilizar estándares de comunicación ligados íntimamente a un proveedor privado único, lo que restringe a un usuario o a una colectividad a ser cliente de este mismo proveedor, reforzando así de manera significativa los fenómenos de abuso de posición dominante.

Los servicios del Estado utilizan a menudo software cuyo código fuente no está disponible, lo que les impide hacer que se corrijan los errores que los propios proveedores se niegan a corregir o verificar si existen defectos de seguridad en las aplicaciones sensibles. Los servicios del Estado utilizan, a veces sin saberlo, software que transmite secretamente información considerada *a priori* como confidencial a sociedades u organismos extranjeros.

Ahora bien, los modelos económicos de la industria del software y de las telecomunicaciones desarrollados por el mercado se basan en gran medida en la apropiación de una clientela y en la valoración exponencial de la captación de los perfiles de los usuarios. Estos modelos económicos favorecen estrategias de incompatibilidad, de secreto industrial, de obsolescencia programada y de violación de las libertades individuales. Si bien el Estado francés no puede pretender eliminar por ley estas tendencias subyacentes debido al carácter transnacional de

ANOTACIONES

las redes de comunicación, sí que puede, sin embargo, favorecer el desarrollo sobre suelo francés de una sociedad de la información que sea respetuosa con las libertades públicas, con la seguridad del consumidor y con la igualdad de oportunidades, y cabe esperar que juegue un papel precursor en Europa y en el mundo.

La ley que proponemos se fundamenta en cinco principios: el libre acceso del ciudadano a la información pública, la perennidad de los datos públicos, la seguridad del Estado, la seguridad del consumidor en la sociedad de la información y el principio de interoperabilidad de la legislación sobre software.

Para garantizar el libre acceso del ciudadano a la información pública, hace falta que la codificación de los datos informáticos comunicados por la Administración no esté ligada a un proveedor único. Los estándares abiertos, es decir, aquellos cuyas reglas de codificación de la información son públicas, permiten garantizar ese libre acceso al permitir, si es necesario, el desarrollo de software libre compatible.

Para garantizar la perennidad de los datos públicos, es necesario que la utilización y mantenimiento del software no dependa de la buena intención de los creadores del software. Hacen falta sistemas cuya evolución esté siempre garantizada mediante la disponibilidad del código fuente. El principio de disponibilidad del código fuente en el marco de los contratos basados en licencias, principio presente a fecha de hoy solamente como opción en la legislación sobre compras públicas de utilidades y paquetes software, debe convertirse en la regla y aplicarse a todas las compras públicas de software.

Hemos desechado voluntariamente una aproximación legislativa basada exclusivamente en el uso de software libre. Sería inoportuno que, cualquiera que fuese la calidad reconocida del software libre, el Estado favoreciese un determinado modelo económico de edición de software. Al contrario, el recurso obligatorio a estándares de comunicación abiertos y la publicación del código fuente garantizan una igualdad de oportunidades conforme al principio de interoperabilidad de la legislación sobre software.

Para garantizar la seguridad nacional, son necesarios sistemas desprovistos de elementos que permitan un control a distancia o la

ANOTACIONES

transmisión involuntaria de información a un tercero. Necesitamos sistemas cuyo código fuente esté accesible libremente al público para permitir el examen por un gran número de expertos mundiales independientes. Nuestro proyecto de ley debería aportar más seguridad al Estado, pues el conocimiento del código fuente eliminará el número creciente de software que contiene "puertas traseras".

Nuestro proyecto de ley debería asimismo reforzar la seguridad del consumidor en la sociedad de la información, al permitir el surgimiento de una oferta de software desprovisto de "puertas traseras", que amenazan el respeto de la vida privada y de las libertades individuales.

Pero para que emerja la igualdad de oportunidades es preciso que se reafirme y refuerce el principio de interoperabilidad introducido en la legislación sobre software y en la legislación sobre la compatibilidad. Ambos derechos están hoy amenazados por los actores que gozan de una posición dominante, quienes ponen trabas al surgimiento de la competencia.

Para garantizar la interoperabilidad del software, es necesario que los derechos de propiedad intelectual o industrial de un creador de software no bloqueen el desarrollo del nuevo software compatible que compita con él. El derecho a la compatibilidad para todos, es decir, el derecho a desarrollar, publicar y utilizar libremente un software original compatible con otro, debe estar garantizado por la ley. Asimismo, el principio de interoperabilidad introducido por el derecho europeo del software debe prevalecer sobre los otros derechos eventuales de propiedad intelectual o industrial. En particular, la existencia de una marca sobre un estándar de comunicaciones o de una patente sobre un proceso industrial necesario para implementar un estándar de comunicaciones, no debería permitir a su poseedor bloquear o limitar la difusión libre de software compatible.

Nuestro proyecto de ley puede aplicarse inmediatamente. En efecto, la mayoría de los editores de software están dispuestos a adoptar estándares de comunicación abiertos como los definidos en París, Boston y Tokio por el World Wide Web Consortium. Existen muchos editores de software propietario que están igualmente dispuestos a proporcionar a la Administración francesa el código fuente de sus

ANOTACIONES

productos. Además, la oferta de software libre alrededor del sistema operativo Linux cubrirá en adelante muchas de las necesidades de la Administración. Sin embargo, las administraciones y colectividades no están suficientemente informadas de la existencia de estándares abiertos, o de la oferta de software publicado con código fuente.

Para facilitar la adopción rápida de los estándares abiertos, es necesario reforzar el papel de la comisión interministerial de soporte técnico para el desarrollo de las tecnologías de la información y la comunicación en la Administración (*Mission interministérielle de soutin technique pour le développement des technologies de l'information et de la communication dans l'administration*), y confiarle la misión de hacer y difundir dentro de la Administración un censo de la oferta de estándares abiertos y software publicado con código fuente. En caso de que no exista un mercado, la MTIC se encargará de desarrollar nuevos estándares o nuevo software publicado con su código fuente. Para llevar a cabo estas nuevas misiones, la MTIC se transformará en la Agencia de Tecnologías de la Información y la Comunicación (ATIC).

Cuando no exista un mercado, la ATIC se encargará de desarrollar nuevos estándares o nuevo software publicado con su código fuente. Para conseguir la igualdad de oportunidades, los desarrollos de software que se produzcan se pondrán en el dominio público; por lo tanto, podrán comercializarse tanto como software propietario como software libre, según elija libremente una licencia u otra el editor. La ATIC se encargará también de evaluar el nivel de interoperabilidad, de perennidad y de seguridad del software comprado por la Administración francesa.

Más genéricamente, los sistemas de comunicación abiertos y la disponibilidad del código fuente son indispensables para garantizar a escala europea la interoperabilidad entre los sistemas de información de las administraciones y de los organismos públicos nacionales, y para evitar que la interconexión entre sistemas dependa solamente de la buena voluntad de los editores de software. La ATIC estará encargada también de participar en los trabajos de cooperación internacional en el dominio de las tecnologías de la información y las comunicaciones, y de favorecer la interoperabilidad con los sis-

ANOTACIONES

temas de información de los demás países miembros de la Unión Europea.

Nuestro proyecto de ley responde a las preocupaciones enumeradas arriba. Nos recuerda que el Estado puede jugar un papel en la economía preservando el interés nacional y europeo, defendiendo al mismo tiempo la economía de mercado. Este proyecto de ley permitirá que Francia se erija en defensora de las libertades dentro de las nuevas tecnologías de la información y las comunicaciones.

Artículos

- Artículo 1

Para todos los intercambios de datos informáticos, la Administración del Estado, las administraciones locales y los organismos públicos, tienen la obligación de emplear estándares de comunicación abiertos, constituidos por reglas y procedimientos públicos de intercambio de datos digitales.

- Artículo 2

La Administración, los organismos públicos y las administraciones públicas territoriales están obligadas a utilizar software cuyo código fuente esté accesible.

- Artículo 3

Toda persona física o jurídica tiene derecho a desarrollar, publicar o utilizar software original compatible con los estándares de comunicación de cualquier otro software.

- Artículo 4

Se creará un organismo público del Estado, denominado Agencia de las Tecnologías de la Información y las Comunicaciones. Este organismo será tutelado por el Ministerio de Industria. La ATIC tiene la misión de informar y aconsejar a los servicios del Estado, a las colec-

ANOTACIONES

tividades y a los organismos públicos acerca de la concepción y la identificación de las necesidades técnicas en materia de tecnologías de la información y las comunicaciones. Identificará las necesidades de los servicios públicos en materia de equipos y software, velará por la armonización de los estándares de comunicación y propondrá las prácticas técnicas al uso. Realizará el inventario por sectores de actividad de los estándares abiertos y del software disponible.

En función de este inventario, apoyará el desarrollo de estándares abiertos y de software publicado con su código fuente, y favorecerá la utilización de éste en el dominio público para paliar las carencias del mercado.

La ATIC favorecerá la interoperabilidad con los sistemas de información de otros países miembros de la Unión Europea y participará en los proyectos de cooperación internacional en el dominio de las tecnologías de la información y las comunicaciones. La ATIC tendrá un interlocutor en cada prefectura.

Las modalidades de funcionamiento de la ATIC se establecerán mediante decreto.

- Artículo 5

Las modalidades de aplicación de la presente ley, así como las condiciones de transición desde la situación actual, se fijarán mediante decreto del Consejo de Estado.

- Artículo 6

Los gastos del Estado resultantes de la aplicación de la presente ley se pagarán con cargo a las partidas fijadas en los artículos 575 y 575A del Código general de Impuestos.

6.4.3. Proyecto de ley de Villanueva y Rodrich (Perú)

Texto literal de gran parte del proyecto de ley número 2485, “Ley de Uso de Software Libre en la Administración Pública” de los congresistas peruanos Edgar Villanueva Núñez y Jacques Rodrich Ackerman.

Lectura complementaria

[Ver bibliografía, apartado 200]

ANOTACIONES

Exposición de motivos

La complejidad del mundo en que vivimos exige permanentemente de los países una constante revisión y adaptación de sus marcos institucionales, por lo que, de este modo consigue mantenerse al compás de los nuevos retos que nos impone el sorprendente desarrollo tecnológico.

El descubrimiento de nuevas tecnologías informáticas, entre ellas la del software libre, se ha configurado con el devenir del tiempo en un instrumento idóneo para asegurar de una manera más idónea la protección de la información con la que cuenta el Estado.

De esta manera, la tecnología cumple su función facilitadora de las diferentes y múltiples actividades humanas, siendo una de ellas el manejo de información reservada en las canteras del Estado.

Según la Constitución Política de Perú, en el inciso 5 del artículo 2, "toda persona tiene derecho a solicitar sin expresión de causa la información que requiera y a recibirla de cualquier entidad pública, en el plazo legal, con el costo que suponga el pedido. Se exceptúan las informaciones que afectan a la intimidad personal y las que expresamente se excluyen por ley o por razones de seguridad nacional".

Del mismo modo, el inciso 6 del mismo artículo enfatiza el derecho de toda persona a "que los servicios informáticos, computarizados o no, públicos o privados, no suministren informaciones que afecten a la intimidad personal y familiar".

En este orden de ideas, es evidente la preocupación que muestra nuestra carta fundamental por establecer reaseguros institucionales que por un lado protejan la libertad de los ciudadanos para acceder a información pública y por el otro, la de guardar la debida reserva de información que afecten a la intimidad personal y familiar, así como razones de seguridad nacional.

La garantía de estos derechos consagrados en nuestra Constitución no pasa únicamente por la buena voluntad de los agentes del Estado para el cumplimiento normativo de la misma, sino también por el

ANOTACIONES

empleo de tecnologías que en unos casos coadyuvan y en otros no a una efectiva protección de dichos derechos ciudadanos.

En este contexto, es imperioso para el Estado incorporar aquellas tecnologías que ayudan a reforzar el ejercicio del derecho de la información de los ciudadanos y su debida reserva en los casos que lo ameritan.

La utilización del software libre en todas las instituciones del Estado apunta en este sentido. Básicamente, podemos decir que los principios elementales que animan al presente proyecto de ley se vinculan a garantías básicas del Estado democrático de derecho y lo podemos resumir en los siguientes:

1. Libre acceso del ciudadano a la información pública
2. Perennidad de los datos públicos
3. Seguridad del Estado y de los ciudadanos

Para garantizar el libre acceso de los ciudadanos a la información pública, resulta indispensable que la codificación de los datos no esté ligada a un único proveedor. El uso de formatos estándar y abiertos permite garantizar este libre acceso, logrando si fuera necesario la creación de software compatible.

Para garantizar la perennidad de los datos públicos, es indispensable que la utilización y el mantenimiento del software no dependan de la buena voluntad de los proveedores, ni de las condiciones monopólicas, impuestas por éstos. Se precisan sistemas cuya evolución pueda ser garantizada gracias a la disponibilidad del código fuente.

Para garantizar la seguridad nacional, resulta indispensable contar con sistemas desprovistos de elementos que permitan el control a distancia o la transmisión no deseada de información a terceros. Por lo tanto, se requieren sistemas cuyo código fuente sea libremente accesible al público para permitir su examen por parte del propio Estado, los ciudadanos y un gran número de expertos independientes en el mundo.

ANOTACIONES

Esta propuesta aporta mayor seguridad, pues el conocimiento del código fuente eliminará el creciente número de programas con código espía.

De igual forma, la iniciativa de ley potencia la seguridad de los ciudadanos, tanto en su condición de titulares legítimos de la información manejada por el Estado, cuanto en su condición de consumidores. En este último caso, permite el surgimiento de una oferta extensa de software libre desprovisto de potencial código espía susceptible de poner en riesgo la vida privada y las libertades individuales.

El Estado, en aras de mejorar la calidad de la gestión pública en tanto custodio y administrador de información privada, establecerá las condiciones en que los organismos estatales adquirirán software en el futuro, es decir de un modo compatible con las garantías constitucionales y principios básicos antes desarrollados.

El proyecto expresa claramente que para que un programa o software cualquiera sea aceptable por el Estado, no basta con que el programa sea técnicamente suficiente para llevar a cabo una tarea, sino que además las condiciones de contratación deben satisfacer una serie de requisitos en materia de licencia, sin los cuales el Estado no puede garantizar al ciudadano el procesamiento adecuado de sus datos, velando por su integridad, confidencialidad y accesibilidad a lo largo del tiempo, aspectos críticos para su desempeño.

El Estado establece condiciones para el empleo del software por parte de las instituciones estatales, sin inmiscuirse de modo alguno en las transacciones del sector privado. Constituye un principio jurídicamente reconocido que el Estado no tiene el amplio espectro de libertad contractual del sector privado, pues precisamente está limitado en sus acciones por el deber de transparencia de los actos públicos, y en este sentido, la tutela del mejor interés común debe tener preeminencia cuando se legisla sobre la materia.

El proyecto, asimismo, garantiza el principio de igualdad ante la ley, pues ninguna persona natural y jurídica está excluida del derecho de proveer estos bienes, en las condiciones fijadas en la pre-

ANOTACIONES

sente iniciativa y sin más limitaciones que las establecidas en la Ley de Contrataciones y Adquisiciones del Estado (TUO Decreto Supremo N 012-2001-PCM).

Adicionalmente a estas ventajas, podemos resaltar una serie de beneficios que como consecuencia de esta medida se manifestarán inmediatamente después de ser ejecutadas.

En primer lugar están las oportunidades de trabajo para programadores locales. Del universo de software1 para servidores que se comercializó en EE.UU. el año pasado, el 27% correspondió a programas "libres", proporción verdaderamente significativa para ese enorme y exigente mercado. La cifra es elocuente y constituye una respuesta contundente a quienes creen que el software libre implicará una fuerte limitación a la ocupación de los programadores del país. Al contrario, la iniciativa permitirá liberar una gran cantidad de recursos, y un incentivo para potenciar la creatividad humana.

Al emplear el software libre, los profesionales pueden analizar a fondo los problemas y mejorar los desarrollos en todos los casos que sea necesario, nutriéndose del software libre disponible globalmente bajo distintas licencias. Constituye un campo ideal para aplicar creatividad, aspecto en el que los jóvenes peruanos alcanzarían buenos desempeños.

Por otro lado, mediante el software libre se elimina el uso de software ilegal que se "pasea" por algunas instituciones del Estado. El uso no permitido de software dentro del Estado o la mera sospecha de ello constituye un poderoso incentivo para cualquier funcionario público para modificar esa situación que atentaría contra los derechos de propiedad intelectual.

Si bien es correcto decir que no es necesaria la adopción de software libre para cumplir con la ley, su empleo generalizado reducirá drásticamente las situaciones irregulares y obrará como *vector de contagio legal*, tanto dentro del Estado como en el ámbito privado.

Son muchos los países que están reconociendo formalmente el uso exclusivo del software libre en el sector estatal.

ANOTACIONES

Entre ellos tenemos a Francia, donde está en discusión una norma legal sobre el tema. El Gobierno de la ciudad de México, D.F. ya ha iniciado una importante migración para la adopción de software libre en forma generalizada, siendo este país líder en Occidente. También en Brasil, el estado de Recife ha decidido su adopción. La República Popular China ha adoptado desde hace varios años el software libre como una política de Estado. Al igual que en los países escandinavos. En EE.UU., la NASA y la US NAVY, entre muchas otras organizaciones, han adoptado software libre para alguna de sus necesidades, entre otras iniciativas gubernamentales y del sector privado.

Finalmente, el proyecto de ley en cuestión otorga a la Presidencia del Consejo de Ministros la ejecución de la presente ley por ser este organismo el que concentra la dirección de todas las instituciones gubernamentales. En este sentido, tiene una ventaja estratégica para realizar la reforma pertinente y el proceso migratorio de software propietario a software libre.

Es en este orden de ideas como se han precisado estos aspectos en la presente proposición legislativa.

Análisis coste-beneficio

La presente iniciativa no genera gasto alguno al erario nacional. Eso sí, para el cumplimiento de sus fines será necesario producir una reasignación del gasto gubernamental, cuya incidencia se circunscribe a lo efectivamente gastado por cada organismo gubernamental en los procesos de contrataciones y licitaciones del Estado para la adquisición de programas informáticos.

Si bien es cierto que el software libre con relación al software propietario representa un ahorro sustancial a la economía del Estado, no es el punto principal de apoyo. Como señalamos antes, su ventaja comparativa se focaliza en los reaseguros tecnológicos que el programa otorga a la información con la que cuenta el Estado, información que en muchos casos es de carácter reservado.

En este sentido, una mejor protección de los derechos ciudadanos constituye un beneficio inconmensurable que debe ser reconocido desde el punto de vista del análisis coste-beneficio.

ANOTACIONES

Podemos resumir los beneficios del proyecto en los siguientes tópicos:

- Seguridad nacional

El Estado para cumplir sus funciones debe almacenar y procesar información relativa a los ciudadanos. La relación entre el individuo y el Estado depende de la privacidad e integridad de estos datos, que deben ser adecuadamente resguardados contra tres riesgos específicos:

- 1. Riesgo de filtración, los datos confidenciales deben ser tratados de tal manera que el acceso a ellos sea posible exclusivamente para las personas e instituciones autorizadas.

- 2. Riesgo de imposibilidad de acceso, los datos deben ser almacenados de tal forma que el acceso a ellos por parte de las personas e instituciones autorizadas esté garantizado durante toda la vida útil de la información.

- 3. Riesgo de manipulación, la modificación de los datos debe estar restringida nuevamente a las personas e instituciones autorizadas.

Con el software libre estos riesgos se atenúan considerablemente.

Permite al usuario la inspección completa y exhaustiva del mecanismo mediante el cual procesa los datos. El hecho de que el programa de software libre permite la inspección del programa es una excelente medida de seguridad, ya que al estar expuestos los mecanismos, éstos están constantemente a la vista de profesionales capacitados, con lo que se vuelve inmensamente más difícil ocultar funciones maliciosas, aun cuando el usuario final no se toma el trabajo de buscarlas él mismo.

- Independencia tecnológica

Con el software propietario no hay libertad de contratación en lo que se refiere a ampliaciones y correcciones del sistema que utiliza, se produce una dependencia tecnológica en la que el proveedor está en condiciones de dictar unilateralmente términos, plazos y precios.

Con el software libre se permite al usuario el control, corrección y modificación del programa para adecuarlo a sus necesidades. Esta libertad no está destinada solamente a los programadores Si bien son éstos los que pueden capitalizarla de primera mano, los usuarios también se benefician enormemente, porque de esta manera pueden contratar a cualquier programador (no necesariamente al autor original) para que corrija errores o añada funcionalidad.

- El desarrollo local

En el caso del software propietario, el usuario está habilitado para ejecutar un programa, pero no para inspeccionarlo ni modificarlo, entonces no puede aprender de él, se vuelve dependiente de una tecnología que no sólo no comprende, sino que le está expresamente vedada. Los profesionales de su entorno, que podrían ayudarlo a alcanzar sus metas, están igualmente limitados: como el funcionamiento del programa es secreto y su inspección está prohibida, no es posible arreglarlo. De esa manera los profesionales locales ven sus posibilidades de ofrecer valor agregado cada vez más limitadas, y sus horizontes laborales se estrechan junto con sus oportunidades de aprender más. Con el software libre se neutralizan enormemente estas desventajas del software propietario.

- El coste del software

Se reduce considerablemente al ser libre, pues no hay necesidad de estar solicitando sistemáticamente las licencias del caso para continuar con la utilización del programa. Esto sucede con el software propietario. Es importante para el usuario poder mantener estos costes bajo control, pues de lo contrario puede llegar a verse impedido para llevar a cabo sus metas, a fuerza de derogaciones no planificadas. He aquí una vez más la dependencia tecnológica que ayuda a enfrentar el software libre.

- Mayores fuentes de trabajo

Con el software libre se libera mano de obra existente en el país que estaba embotellada como consecuencia de la dependencia tecnológica del software propietario. Ahora se asignarían recursos de los usuarios (en este caso las instituciones del Estado) para mantenimiento y soporte informático del software libre.

ANOTACIONES

- Fomento de la creatividad e iniciativa empresarial

Costes

El gran coste que supone el cambio de software propietario a software libre se circunscribe al proceso migratorio. Si bien es cierto que el proceso migratorio involucra costes en relevamientos, toma de decisiones para implementar los nuevos sistemas, mano de obra para implementar el cambio, conversión de datos, reentrenamiento del personal y, eventualmente, gastos en licencias y/o desarrollo y tiempo; no es menos cierto que todos estos costes son fijos y se pagan por única vez; en cambio, el software propietario en funcionamiento ahora, también tiene sus costes fijos que fueron pagados y no pueden ser recuperados. Pero, además de estos costes hay otros involucrados en el software propietario: actualizaciones permanentes (a veces acentuada por un monopolio autosostenido) y sobre todo el inmenso precio que tiene para el Estado la pérdida de las libertades, que le garantizan el control de su propia información. Estos costes son permanentes y crecientes a lo largo del tiempo, y tarde o temprano superan a los costes fijos de realizar una migración.

En fin, son mayores los beneficios que los costes que supone el proceso de migración.

Fórmula legal

Artículo 1. Objeto de la ley

En todas las instituciones del Estado se emplea el uso exclusivo de programas o software libres en sus sistemas y equipamientos de informática.

Artículo 2. Ámbito de aplicación

Los poderes Ejecutivo, Legislativo y Judicial, los organismos autónomos y descentralizados, sean regionales o locales, y las empresas donde el Estado posea mayoría accionaria, harán uso de programas o software libres en sus sistemas y equipamientos de informática.

ANOTACIONES

Artículo 3. Autoridad de aplicación

La autoridad encargada de poner en ejecución la presente ley será la Presidencia del Consejo de Ministros.

Artículo 4. Definición de software libre

Para los efectos de la presente ley se entenderá por programa o software libre aquel cuya licencia de uso garantice al usuario, sin coste adicional, las siguientes facultades:

- 1. Uso irrestricto del programa para cualquier propósito.
- 2. Acceso irrestricto al código fuente o de origen respectivo.
- 3. Inspección exhaustiva de los mecanismos de funcionamiento del programa.
- 4. Uso de los mecanismos internos y de porciones arbitrarias del programa para adaptarlos a las necesidades del usuario.
- 5. Confección y distribución libre de copias del programa.
- 6. Modificación del programa y distribución libre, tanto de las alteraciones como del nuevo programa resultante, bajo las mismas condiciones del programa original.

Artículo 5. Excepciones

En caso de no existir una solución que utilice software libre y permita satisfacer una necesidad determinada, las instituciones del Estado podrán adoptar las siguientes alternativas, atendiendo al orden de prelación siguiente:

Si median exigencias de tiempo verificables para atender un problema técnico y se halla disponible en el mercado software propietario, el organismo que lo demande podrá gestionar ante la autoridad de

ANOTACIONES

aplicación un permiso de excepción de utilización de software propietario que reúna las siguientes características:

- 1. Se seleccionará en primer término los programas que cumplan con todos los criterios enunciados en el artículo 4 de la presente ley, excepto por la facultad de distribución del programa modificado. En este caso, el permiso de excepción podrá ser definitivo.

- 2. Si no se puede disponer de programas de la categoría precedente, se deberán escoger aquéllos para los que exista un proyecto de desarrollo avanzado de tipo libre. En este caso, el permiso de excepción será transitorio y caducará automáticamente cuando el software libre pase a estar disponible con la funcionalidad que sea necesaria.

- 3. Si no se encuentran productos de estas condiciones, se podrá optar por programas propietarios, pero el permiso de excepción emanado de la autoridad de aplicación caducará automáticamente a los dos años de haberse emitido, debiendo ser renovado previa constatación de que no exista disponible en el mercado una solución de software libre satisfactoria.

La autoridad de aplicación otorgará un permiso de excepción únicamente si el organismo estatal solicitante garantiza el almacenamiento de los datos en formatos abiertos, sin perjuicio del pago de las licencias propietarias respectivas.

Artículo 6. Permisos educativos

Toda entidad educativa dependiente del Estado está habilitada para gestionar un permiso de software propietario para su uso en investigación, previo pago de los derechos de autor correspondientes y las licencias del caso, siempre que el objeto de investigación esté directamente asociado al uso del programa en cuestión.

Artículo 7. Transparencia de las excepciones

Las excepciones emanadas de la autoridad de aplicación deberán ser sustentadas y publicadas en la página web del portal del Estado.

ANOTACIONES

La resolución que autoriza la excepción deberá enumerar los requisitos funcionales concretos que el programa debe satisfacer.

Artículo 8. Autorización excepcional

En el caso de que alguna entidad del Estado comprendido en el artículo 2.° de la presente ley sea autorizada excepcionalmente para adquirir software propietario para almacenar o procesar datos, cuya reserva sea necesario preservar, la autoridad de aplicación deberá publicar en el portal del Estado un informe, donde se expliquen los riesgos asociados con el uso de software de dichas características para esa aplicación en particular.

Los permisos de excepción otorgados a los organismos del Estado relacionados con la seguridad y la defensa nacional están exceptuados de la obligación anteriormente expuesta.

Artículo 9. Responsabilidades

La máxima autoridad administrativa y autoridad técnica e informática de cada institución del Estado asumen la responsabilidad por el cumplimiento de esta ley.

Artículo 10. Norma reglamentaria

El poder ejecutivo reglamentará en un plazo de ciento ochenta días, las condiciones, tiempos y formas en que se efectuará la transición de la situación actual a una que satisfaga las condiciones de la presente Ley y orientará, en tal sentido, las licitaciones y contrataciones futuras de software realizadas a cualquier título.

Asimismo, se encargará de dirigir el proceso migratorio del sistema de software propietario a libre, en todos los casos que las circunstancias lo exijan.

Artículo 11. Glosario de términos

- a. Programa o "software". Cualquier secuencia de instrucciones usada por un dispositivo de procesamiento digital de datos para

ANOTACIONES

llevar a cabo una tarea específica o resolver un problema determinado.

- b. Ejecución o empleo de un programa. Acto de utilizarlo sobre cualquier dispositivo de procesamiento digital de datos para realizar una función.

- c. Usuario. Aquella persona física o jurídica que emplea el software.

- d. Código fuente o de origen, o programa fuente o de origen. Conjunto completo de instrucciones y archivos digitales originales creados o modificados por quien los programa, más todos los archivos digitales de soporte, como tablas de datos, imágenes, especificaciones, documentación, y cualquier otro elemento que sea necesario para producir el programa ejecutable a partir de ellos. Como excepción, podrán excluirse de este conjunto aquellas herramientas y programas que sean habitualmente distribuidos como software libre por otros medios, como, entre otros, compiladores, sistemas operativos y librerías.

- e. Programa o software libre. Aquel cuyo empleo garantice al usuario, sin coste adicional, las facultades siguientes:

  - e.1. Ejecución irrestricta del programa para cualquier propósito.

  - e.2. Acceso irrestricto al código fuente o de origen respectivo.

  - e.3. Inspección exhaustiva de los mecanismos de funcionamiento del programa.

  - e.4. Uso de los mecanismos internos y de cualquier porción arbitraria del programa para adaptarlo a las necesidades del usuario.

  - e.5. Confección y distribución pública de copias del programa.

  - e.6. Modificación del programa y distribución libre, tanto de las alteraciones como del nuevo programa resultante, bajo las mismas condiciones del programa original.

ANOTACIONES

- f. Software propietario (programa no libre). Aquel que no reúne todos los requisitos señalados en el término precedente.

- g. Formato abierto. Cualquier modo de codificación de información digital que satisface tanto los estándares existentes, como las condiciones siguientes, que:

  - g.1. su documentación técnica completa esté disponible públicamente.

  - g.2. el código fuente de al menos una implementación de referencia completa esté disponible públicamente.

  - g.3. no existan restricciones para la confección de programas que almacenan, transmiten, reciben o acceden a datos codificados de esta manera.

6.4.4. Cartas de Microsoft Perú y del congresista Villanueva

El 21 de marzo del 2002, Juan Alberto González, como gerente general de Microsoft Perú, envió una carta al congresista Edgar Villanueva Núñez, con motivo de su proyecto de ley sobre software libre. El 8 de abril el congresista respondió. Aquí se incluyen los textos literales y casi completos de ambas cartas (se excluyen los párrafos no relacionados con el proyecto de ley).

Lectura complementaria

[Ver bibliografía, apartados 107 y 159]

Carta de Microsoft Perú

De otro lado, como quedamos en esta reunión, nosotros asistimos al Foro realizado en el Congreso de la República el 6 de marzo, a propósito del proyecto de ley que Usted lidera, en donde pudimos escuchar las diferentes presentaciones que hoy nos llevan a exponer nuestra posición a fin de que Usted tenga un panorama más amplio de la real situación.

El proyecto establece la obligatoriedad de que todo organismo público debe emplear exclusivamente software libre, es decir de código

ANOTACIONES

abierto, lo cual transgrede los principios de la igualdad ante la ley, el de no discriminación y los derechos a la libre iniciativa privada, libertad de industria y contratación protegidos en la constitución.

El proyecto, al hacer obligatorio el uso de software de código abierto, establecería un tratamiento discriminatorio y no competitivo en la contratación y adquisición de los organismos públicos contraviniendo los principios de base de la Ley 26850 de Contrataciones y Adquisiciones del Estado.

Así, al obligar al Estado a favorecer un modelo de negocios que apoyaría exclusivamente el software de código abierto, el proyecto sólo estaría desalentando a las compañías fabricantes locales e internacionales que son las que verdaderamente realizan importantes inversiones, crean un significativo número de puestos de empleos directos e indirectos, además de contribuir al PBI vs. un modelo de software de código abierto que tiende a tener un impacto económico cada vez menor debido a que crea principalmente empleos en servicio.

El proyecto de ley impone el uso de software de código abierto sin considerar los peligros que esto pueda conllevar desde el punto de vista de seguridad, garantía y posible violación de los derechos de propiedad intelectual de terceros.

El proyecto maneja de manera errónea los conceptos de software de código abierto, que no necesariamente implica que sea software libre o de coste cero, llegando a realizar conclusiones equívocas sobre ahorros para el Estado, sin ningún sustento coste-beneficio que valide la posición.

Es equivocado pensar que el software de código abierto es gratuito. Investigaciones realizadas por Gartner Group (importante investigadora del mercado tecnológico reconocida a nivel mundial) han señalado que el coste de adquisición del software (sistema operativo y aplicaciones) se reduce a sólo el 8% del total de costes que las empresas e instituciones deben asumir como consecuencia del uso racional y realmente provechoso de la tecnología. El otro 92% lo constituyen: costes de implantación, capacitación, soporte, mantenimiento, administración e inoperatividad.

ANOTACIONES

Uno de los argumentos que sustentan el proyecto de ley es la supuesta gratuidad del software de código abierto, comparado con los costes del software comercial, sin tener en cuenta que existen modalidades de licenciamiento por volumen que pueden ser sumamente ventajosas para el Estado, tal como se ha logrado en otros países.

Adicionalmente, la alternativa adoptada por el proyecto (i) es claramente más costosa por los altos costes que supone una migración y (ii) pone en riesgo la compatibilidad y posibilidad de interoperabilidad de las plataformas informáticas dentro del Estado, y entre el Estado y el sector privado, dada la centena de versiones que existen de software de código abierto en el mercado.

El software de código abierto en su mayoría no ofrece los niveles de servicio adecuados ni la garantía de fabricantes reconocidos para lograr mayor productividad por parte de los usuarios, lo cual ha motivado que diferentes entidades públicas hayan retrocedido en su decisión de ir por una solución de software de código abierto y se encuentren utilizando software comercial en su lugar.

El proyecto desincentiva la creatividad de la industria peruana de software, que factura US$ 40 millones/año, exporta US$ 4 millones (10.º en el ranking de productos de exportación no tradicional, más que artesanías) y es una fuente de empleo altamente calificado. Con una ley que incentive el uso de software de código abierto, los programadores de software pierden sus derechos de propiedad intelectual y su principal fuente de retribución.

El software de código abierto, al poder ser distribuido gratuitamente, tampoco permite generar ingresos para sus desarrolladores por medio de la exportación. De este modo, se debilita el efecto multiplicador de la venta de software a otros países y por lo tanto el crecimiento de esta industria, cuando contrariamente las normas de un gobierno deben estimular la industria local.

En el foro se discutió sobre la importancia del uso de software de código abierto en la educación, sin comentar el rotundo fracaso de esta iniciativa en un país como México, donde precisamente los funcionarios del Estado que fundamentaron el proyecto, hoy expresan que el

ANOTACIONES

software de código abierto no permitió brindar una experiencia de aprendizaje a alumnos en la escuela, no se contó con los niveles de capacitación a nivel nacional para dar soporte adecuado a la plataforma, y el software no contó y no cuenta con los niveles de integración para la plataforma que existe en las escuelas.

Si el software de código abierto satisface todos lo requerimientos de las entidades del Estado ¿por qué se requiere una ley para adoptarlo? ¿No debería ser el mercado el que decida libremente cuáles son los productos que le dan más beneficios o valor?

Respuesta del congresista Villanueva

Ante todo, agradezco su carta del 25 de marzo del 2002 donde manifiesta la posición oficial de Microsoft respecto al Proyecto de Ley N. 1609, Software Libre en la Administración Pública, que sin duda se halla inspirada en el deseo de que el Perú logre situarse adecuadamente en el contexto tecnológico global. Animado de ese mismo espíritu y convencido de que a través del intercambio de ideas claras y abiertas hemos de encontrar las mejores soluciones, me permito contestar mediante la presente los comentarios incluidos en su carta.

Sin dejar de reconocer que opiniones como la suya constituyen un aporte significativo, me hubiese resultado aún más valioso si, además de formular objeciones de índole general (que luego analizaremos en detalle) hubiera agregado argumentos sólidos sobre las ventajas que el software propietario puede reportar al Estado peruano y a sus ciudadanos en general, pues ello habría permitido un intercambio a todas luces más esclarecedor respecto de cada una de nuestras posiciones.

Con el objetivo de ordenar el debate, asumiremos que lo que Ud. llama *software de código abierto* es lo que el proyecto define como *software libre*, puesto que existe software cuyo código es distribuido junto con los programas, pero no encaja en la definición establecida en el proyecto; y lo que Ud. llama *software comercial* es lo que el proyecto define como *propietario* o *no libre*, puesto que existe software libre que se comercializa en el mercado por un precio como cualquier otro bien o servicio.

ANOTACIONES

También es preciso dejar en claro que el propósito del proyecto al que nos referimos no está directamente relacionado con la cantidad de ahorro directo que pueda obtenerse por el empleo de software libre en las instituciones estatales. Éste es, en todo caso, un valor agregado marginal, pero de ninguna manera el foco del objetivo del proyecto. Los principios elementales que animan al proyecto se vinculan a las garantías básicas de un Estado democrático de derecho, como:

- Libre acceso del ciudadano a la información pública.
- Perennidad de los datos públicos.
- Seguridad del Estado y de los ciudadanos.

Para garantizar el libre acceso de los ciudadanos a la información pública, resulta indispensable que la codificación de los datos no esté ligada a un único proveedor. El uso de formatos estándar y abiertos permite garantizar este libre acceso, logrando si fuera necesario la creación de software libre compatible.

Para garantizar la perennidad de los datos públicos, es indispensable que la utilización y el mantenimiento del software no dependan de la buena voluntad de los proveedores, ni de las condiciones monopólicas impuestas por éstos. Por esta razón, el Estado necesita sistemas cuya evolución pueda ser garantizada gracias a la disponibilidad del código fuente.

Para garantizar la seguridad del Estado o seguridad nacional, resulta indispensable contar con sistemas desprovistos de elementos que permitan el control a distancia o la transmisión no deseada de información a terceros. Por lo tanto, se requieren sistemas cuyo código fuente sea libremente accesible al público para permitir su examen por parte del propio Estado, los ciudadanos, y un gran número de expertos independientes en el mundo. Nuestra propuesta aporta mayor seguridad, pues el conocimiento del código fuente eliminará el creciente número de programas con *código espía*.

Asimismo, nuestra propuesta refuerza la seguridad de los ciudadanos, tanto en su condición de titulares legítimos de la información manejada por el Estado, cuanto en su condición de consumidores. En este ultimo caso, al permitir el surgimiento de una oferta extensa

ANOTACIONES

de software libre desprovisto de potencial *código espía* susceptible de poner en riesgo la vida privada y las libertades individuales.

En este sentido, el proyecto de ley se limita a establecer las condiciones en que los organismos estatales adquirirán software en el futuro, es decir, de un modo compatible con la garantía de esos principios básicos.

De la lectura del proyecto quedará claro que una vez aprobada:

- la ley no prohíbe la producción de software propietario
- la ley no prohíbe el comercio de software propietario
- la ley no dicta qué software concreto usar
- la ley no dicta a qué proveedor se compra el software
- la ley no limita los términos en que se puede licenciar un producto de software

Lo que el proyecto expresa claramente es que el software, para ser aceptable para el Estado, no basta con que sea técnicamente suficiente para llevar a cabo una tarea, sino que además las condiciones de contratación deben satisfacer una serie de requisitos en materia de licencia, sin los cuales el Estado no puede garantizar al ciudadano el procesamiento adecuado de sus datos, velando por su integridad, confidencialidad y accesibilidad a lo largo del tiempo, porque son aspectos muy críticos para su normal desempeño.

Estamos de acuerdo, Sr. González, en el hecho de que la tecnología de información y comunicaciones tiene un impacto en la calidad de vida de los ciudadanos significativo (sin que por ello sea siempre positivo o de efecto neutro). También coincidiremos seguramente, en que los valores básicos que he señalado arriba son fundamentales en una nación democrática como el Perú. Desde luego estamos muy interesados en conocer cualquier forma alternativa de garantizar estos principios, que no sea la de recurrir al empleo de software libre en los términos definidos en el proyecto de ley.

ANOTACIONES

En cuanto a las observaciones que Ud. formula, pasaremos ahora a analizarlas en detalle:

En primer lugar, señala que: "1. El proyecto establece la obligatoriedad de que todo organismo público debe emplear exclusivamente software libre, es decir de código abierto, lo cual transgrede los principios de la igualdad ante la ley, el de no discriminación y los derechos a la libre iniciativa privada, libertad de industria y contratación protegidos en la constitución.".

Esta apreciación constituye un error. De ningún modo el proyecto afecta a los derechos que Ud. enumera; sólo se limita a establecer condiciones para el empleo del software por parte de las instituciones estatales, sin inmiscuirse en modo alguno en las transacciones del sector privado. Es un principio bien establecido que el Estado no tiene el amplio espectro de libertad contractual del sector privado, pues precisamente está limitado en sus acciones por el deber de transparencia de los actos públicos; y en ese sentido, la preservación del mejor interés común debe prevalecer cuando se legisla sobre la materia.

El proyecto protege la igualdad ante la ley, pues ninguna persona natural o jurídica está excluida del derecho de ofrecer estos bienes al Estado en las condiciones fijadas en el proyecto y sin más limitaciones que las establecidas en la Ley de Contrataciones y Adquisiciones del Estado (TUO por Decreto Supremo No. 012-2001-PCM).

El proyecto no introduce discriminación alguna, pues sólo establece *cómo* han de proveerse estos bienes (lo cual es una potestad estatal) y no *quién* ha de proveerlos (lo que en efecto resultaría discriminatorio si se impusieran restricciones basadas en origen nacional, raza, religión, ideología, preferencia sexual, etc.) Por el contrario, el proyecto es decididamente antidiscriminatorio. Es así porque al determinar sin lugar a dudas las condiciones de provisión del software, impide a los organismos estatales el uso de programas cuyo licenciamiento incluya condiciones discriminatorias.

Resulta obvio por lo expuesto en los dos párrafos previos, que el proyecto no atenta contra la libre iniciativa privada, pues ésta puede elegir siempre bajo qué condiciones producirá el software; algunas de

ANOTACIONES

éstas serán aceptables para el Estado, y otras no lo serán porque contrarían la garantía de los principios básicos enumerados arriba. Esta libre iniciativa es desde luego, compatible con la libertad de industria y con la libertad de contratación (en los términos acotados en que el Estado puede ejercer esta última). Cualquier sujeto privado puede producir software en las condiciones que el Estado lo requiere, o puede abstenerse de hacerlo. Nadie está forzado a adoptar un modelo de producción, pero si desea proveer software al Estado, deberá proporcionar los mecanismos que garantizan los principios básicos, y que son los manifestados en el proyecto.

A manera de ejemplo: nada en el texto del proyecto impediría a su empresa ofrecer a los organismos del Estado su *suite* de oficina, en las condiciones definidas en el proyecto y fijando el precio que ustedes consideren conveniente. Si no lo hiciera, no se deberá a restricciones impuestas por la ley, sino a decisiones empresariales respecto al modo de comercializar sus productos, decisiones en las que el Estado no tiene participación.

A continuación señala Ud. que: "2. El proyecto, al hacer obligatorio el uso de software de código abierto, establecería un tratamiento discriminatorio y no competitivo en la contratación y adquisición de los organismos públicos [...]."

Esta afirmación no es sino una reiteración de la anterior, y por ende se encuentra contestada líneas arriba. Pero detengámonos un instante en su apreciación sobre el "tratamiento [...] no competitivo."

Por cierto, al definir cualquier tipo de adquisición, el comprador fija condiciones que se relacionan con el uso propuesto del bien o servicio. Desde luego ello excluye a ciertos fabricantes de la posibilidad de competir, pero no los excluye *a priori*, sino sobre la base de una serie de principios decididos por la voluntad autónoma del comprador, en tanto el proceso se lleve a cabo conforme a la ley. Y en el proyecto se establece que *nadie* está excluido de competir en tanto garantice el cumplimiento de los principios básicos.

Además, el proyecto *estimula* la competencia, pues alienta a generar oferta de software con mejores condiciones de usabilidad, y a optimizar trabajos ya establecidos, en un modelo de mejora constante.

ANOTACIONES

De otro lado, el aspecto central de la competitividad es la oportunidad de proporcionar al consumidor mejores opciones. Ahora bien, es imposible desconocer que el marketing no juega un papel neutral a la hora de presentar la oferta al mercado (pues admitir lo contrario habilitaría a suponer que las inversiones que las empresas realizan en marketing carecen de sentido), y por consiguiente un gasto significativo en este rubro puede influir en las decisiones del comprador. Esta influencia del marketing queda en buena medida mitigada por el proyecto que propulsamos, pues la elección dentro del marco propuesto recae en el *mérito técnico* del producto y no en el esfuerzo de comercialización del productor; en este sentido, la competitividad se acentúa, pues el más pequeño productor de software puede competir en un pie de igualdad con la más poderosa de las corporaciones.

Es necesario recalcar que no hay posición más anti-competitiva que la de los grandes productores de software propietario, que frecuentemente abusan de su posición dominante, porque en innumerables casos proponen como soluciones a problemas planteados por los usuarios: "actualice su software a la nueva versión" (con cargo para el usuario, por supuesto); además, son comunes las interrupciones arbitrarias de asistencia técnica para productos que al sólo juicio del proveedor, son *antiguos*; luego, para recibir algún grado de asistencia técnica, el usuario se ve obligado a migrar (con coste no trivial, especialmente porque suele involucrar cambios de la plataforma de hardware) a nuevas versiones. Y como toda la infraestructura está consolidada en formatos de datos propietarios, el usuario queda *atrapado* en la necesidad de continuar empleando los productos del mismo proveedor, o realizar el enorme esfuerzo de cambiar a otro ambiente (también probablemente propietario).

Agrega Ud.: "3. Así, al obligar al Estado a favorecer un modelo de negocios que apoyaría exclusivamente el software de código abierto, el proyecto sólo estaría desalentando a las compañías fabricantes locales e internacionales que son las que verdaderamente realizan importantes inversiones, crean un significativo número de puestos de empleos directos e indirectos, además de contribuir al PBI vs. un modelo de software de código abierto que tiende a tener un impacto económico cada vez menor debido a que crea principalmente empleos en servicio."

ANOTACIONES

No estoy de acuerdo con lo que Ud. afirma. En parte por lo que Ud. mismo señala en el párrafo 6 de su carta, respecto del peso relativo de los servicios en el contexto del uso de software. Esta contradicción, de por sí, invalidaría su postura. El modelo de servicios, adoptado por gran número de corporaciones en la industria informática, es mucho más significativo, en términos económicos y con tendencia creciente, que el licenciamiento de programas.

Por otra parte, el sector privado de la economía tiene la más amplia libertad para elegir el modelo económico que más convenga a sus intereses, aunque esta libertad de elección quede muchas veces oscurecida de manera subliminal por las desproporcionadas inversiones en marketing de los productores de software propietario.

Adicionalmente, de la lectura de su opinión se desprendería que el mercado estatal es crucial e imprescindible para la industria del software propietario, a tal punto que la opción que el Estado establece en este proyecto eliminaría completamente del mercado a estas empresas. Si es así, deducimos que el Estado estaría subsidiando a la industria del software propietario. En el supuesto negado que esto fuese cierto, entonces el Estado tendría el derecho a aplicar los subsidios al área que considere de mayor valor social; resultaría innegable, en esta improbable hipótesis, que si el Estado decide subsidiar software, debería hacerlo escogiendo el libre por encima del propietario, atendiendo a su efecto social y al uso racional de los dineros de los contribuyentes.

Respecto de los puestos de trabajo generados por el software propietario en países como el nuestro, éstos tratan mayoritariamente tareas técnicas de poco valor agregado; a nivel local, los técnicos que prestan soporte a software propietario producido por empresas transnacionales no están en condiciones de solucionar un *bug*, no necesariamente por falta capacidad técnica o talento, sino porque no disponen del código fuente a reparar. Con software libre se crea empleo técnicamente más calificado y se genera un marco de libre competencia donde el éxito esta sólo vinculado a la capacidad de brindar buen soporte técnico y calidad de servicio, se estimula el mercado y se incrementa el patrimonio común del conocimiento, abriendo alternativas para generar servicios de mayor valor agrega-

do y mejor perfil de calidad beneficiando a todos los actores: productores, prestadores de servicios y consumidores.

Es un fenómeno común en los países en vías de desarrollo que las industrias locales de software obtienen la mayoría de sus ingresos en el área de servicios, o en la construcción de software *ad hoc*. Por lo tanto, cualquier impacto negativo que la aplicación del proyecto pueda tener en este sector se verá compensado con creces por un aumento de la demanda de servicios (a condición de que éstos sean prestados conforme a altos estándares de calidad). Desde luego, es probable que las empresas transnacionales de software, si deciden no competir conforme a estas reglas de juego, sufran alguna disminución de ingresos en términos de facturación por licenciamiento; pero, considerando que estas empresas alegan sostenidamente que mucho del software empleado por el Estado fue copiado ilegalmente, se verá que el impacto no ha de ser extremadamente serio. Ciertamente, en todo caso su fortuna estará determinada por leyes del mercado, cuyos cambios no es posible evitar; muchas empresas tradicionalmente asociadas con el software propietario ya han emprendido un camino firme (apoyado por cuantiosas inversiones) para prestar servicios asociados con el software libre, lo cual demuestra que los modelos no son mutuamente excluyentes.

Con este proyecto el Estado está decidiendo que requiere preservar ciertos valores fundamentales. Y lo decide sobre la base de sus potestades soberanas, sin afectar con ello a ninguna de las garantías constitucionales. Si estos valores pueden ser garantizados sin tener que escoger un modelo económico dado, los efectos de la ley serían aún más beneficiosos. En todo caso, debe quedar claro que el Estado no elige un modelo económico; si sucediera que existe un sólo modelo económico capaz de proveer software tal que satisfaga las garantías básicas de estos principios, se trataría de una circunstancia histórica y no de una decisión arbitraria en favor de un modelo dado.

Prosigue su carta: "4. El proyecto de ley impone el uso de software de código abierto sin considerar los peligros que esto pueda conllevar desde el punto de vista de seguridad, garantía y posible violación de los derechos de propiedad intelectual de terceros."

ANOTACIONES

Aludir de forma abstracta “los peligros que pueda conllevar”, sin especificar siquiera una sola instancia de esos supuestos peligros, denota cuando menos un desconocimiento del tema. Así, pues, permítame ilustrarlo sobre estos puntos.

Sobre seguridad:

En términos generales respecto la seguridad nacional, ya se mencionó inicialmente en los principios básicos del proyecto. En términos más puntuales respecto de la seguridad del software en sí, es bien sabido que el software (propietario o libre) contiene errores de programación o *bugs* (en la jerga informática) en sus líneas de código. Pero también es público y notorio que los *bugs* en el software libre son menos, y se reparan mucho más rápidamente, que en el software propietario. No en vano numerosas organismos públicos responsables por la seguridad informática de los sistemas estatales en países desarrollados prescriben el uso de software libre a iguales condiciones de seguridad y eficiencia.

Lo que resulta imposible probar es que el software propietario sea más seguro que el libre, salvo mediante el escrutinio público y abierto de la comunidad científica y los usuarios en general. Esta demostración es imposible porque el propio modelo del software propietario impide este análisis, con lo que la garantía de seguridad se basa en la palabra bienintencionada (pero a todas luces parcial) del propio productor o sus contratistas.

Corresponde recordar que, en numerosos casos, las condiciones de licenciamiento incluyen cláusulas de *non-disclosure,* que impiden a los usuarios revelar abiertamente las fallas de seguridad halladas en el producto propietario licenciado.

Respecto a garantía:

Como Ud. sabe perfectamente, o podrá determinar leyendo el *End User License Agreement* de los productos que licencia, en la amplísima mayoría de los casos, las garantías están limitadas a la reposición del medio de almacenamiento si éste fuera defectuoso, pero en ningún caso se prevén compensaciones por daños directos o indirectos, lucro cesante, etc. Si como consecuencia de un *bug* de seguridad

ANOTACIONES

en alguno de sus productos, no oportunamente reparado por Uds., un atacante comprometiera sistemas cruciales para el Estado: ¿qué garantías, reparaciones y compensaciones proporcionaría su empresa de acuerdo con sus condiciones de licenciamiento? Las garantías del software propietario, en tanto los programas se entregan “AS IS”, es decir, en el estado en que se encuentran, sin ninguna responsabilidad adicional para el proveedor respecto a su funcionalidad, no difieren en modo alguno de las habituales en el software libre.

Sobre la propiedad intelectual:

Las cuestiones de propiedad intelectual están fuera del ámbito en este proyecto, pues se encuentran amparadas por otras leyes específicas. El modelo de software libre no implica en modo alguno desconocer estas leyes y de hecho, la amplísima mayoría del software libre está amparado por el *copyright*. En realidad, la sola inclusión de esta cuestión en sus observaciones demuestra su confusión respecto del marco legal en que se desenvuelve el software libre. La incorporación de propiedad intelectual ajena en obras que luego se atribuyen como propias no es una práctica de la que se tenga registro en la comunidad del software libre; sí que lo es, lamentablemente, en el terreno del software propietario. Valga a titulo de ejemplo la condena de la Corte Comercial de Nanterre, Francia, del pasado 27 de septiembre del 2001 a Microsoft Corp., por 3 millones de francos en concepto de daños e intereses, por violación de la propiedad intelectual (piratería, según el desafortunado término que su empresa suele usar en su publicidad).

Prosigue diciendo que: “5. El proyecto maneja de manera errónea los conceptos de software de código abierto, que no necesariamente implica que sea software libre o de coste cero, llegando a realizar conclusiones equívocas sobre ahorros para el Estado, sin ningún sustento coste-beneficio que valide la posición.”

Esta observación no es así, en principio la gratuidad y la libertad son conceptos ortogonales: hay software propietario y oneroso (por ejemplo, MS Office), software propietario y gratuito (MS Internet Explorer), software libre y oneroso (distribuciones RedHat, SuSE, etc. del sistema GNU/Linux), software libre y gratuito (Apache, OpenOffice,

ANOTACIONES

Mozilla), y aun software que se licencia bajo diferentes modalidades (MySQL).

Ciertamente que el software libre no es necesariamente gratuito. Y tampoco se desprende del texto del proyecto que deba serlo como bien habrá notado después de leer la norma propuesta. Las definiciones incluidas en el proyecto determinan claramente *qué* debe considerarse software libre, en ningún momento se refieren a la gratuidad. Si bien se mencionan las posibilidades de ahorro en términos de lo pagado por licencias de software propietario, los fundamentos del proyecto hacen clara mención a las garantías fundamentales que se pretende preservar y al estimulo del desarrollo tecnológico local. Puesto que un Estado democrático debe sostener estos principios, no le queda otra solución que emplear software, cuyo código fuente está públicamente disponible e intercambiar información sólo en formatos estándares.

Si el Estado no empleara software con esas características, estaría vulnerando principios republicanos básicos. Por fortuna, además, el software libre implica menores costes totales; pero aun en la hipótesis (fácilmente negada) de que costara más que el propietario, la sola existencia de una herramienta de software libre eficaz para una determinada función informática obligaría al Estado a usarla; no por imperio de este proyecto de ley, sino por los principios elementales que enumeramos al comienzo y que surgen de la esencia misma del Estado democrático de derecho.

Sigue Ud.: "6. Es equivocado pensar que el software de código abierto es gratuito. Investigaciones realizadas por Gartner Group (importante investigadora del mercado tecnológico reconocida a nivel mundial) han señalado que el coste de adquisición del software (sistema operativo y aplicaciones) se reduce a sólo el 8% del total de costes que las empresas e instituciones deben asumir como consecuencia del uso racional y realmente provechoso de la tecnología. El otro 92% lo constituyen: costes de implantación, capacitación, soporte, mantenimiento, administración e inoperatividad."

Este argumento repite lo ya señalado en el párrafo 5 y en parte se contradice con el párrafo 3. Por lo tanto nos remitiremos a lo allí dicho en homenaje a la brevedad. No obstante, permítame señalarle

ANOTACIONES

que incurre en una conclusión falsa en el plano lógico: que el coste de software según Gartner Group sea sólo el 8% en promedio del coste total de utilización, no invalida en forma alguna la existencia de software gratuito, esto es, aquél cuyo costo de licenciamiento es cero.

Además, en este párrafo Ud. indica acertadamente que los componentes de servicio y las pérdidas por indisponibilidad conforman la parte sustancial del costo total de utilización de software; lo que, advertirá, entra en contradicción con su afirmación del valor mínimo de los servicios sugerido en el párrafo 3. Ahora bien, el empleo de software libre contribuye significativamente a disminuir los restantes costes del ciclo de vida. Esta reducción del impacto económico de despliegue, soporte, etc. se registra en varios campos; por un lado, el modelo competitivo de servicios del software libre, cuyo soporte y mantenimiento es posible contratar libremente entre una oferta variada que compite en función de la calidad y el menor coste. Esto es válido para la implantación, la capacitación y el soporte, y en buena medida para el mantenimiento. En segundo lugar, por la característica reproductiva del modelo, hace que el mantenimiento que se realizó en una aplicación sea replicable muy fácilmente, sin incurrir en mayores costes (es decir, sin pagar más de una vez por lo mismo) pues las modificaciones, si así se desea, quedan incorporadas al patrimonio común del conocimiento. En tercero, porque el enorme coste causado por la inoperatividad (*pantallas azules de la muerte*, código malicioso como virus, *worms* y troyanos, excepciones, fallas generales de protección y otros tantos males conocidos) se reduce significativamente al emplear software más estable; y es bien sabido que una de las virtudes más destacables del software libre es su estabilidad.

Afirma luego que: "7. Uno de los argumentos que sustentan el proyecto de ley es la supuesta gratuidad del software de código abierto, comparado con los costes del software comercial, sin tener en cuenta que existen modalidades de licenciamiento por volumen que pueden ser sumamente ventajosas para el Estado, tal como se ha logrado en otros países."

He puntualizado ya que lo que está en cuestión no es el costo del software, sino los principios de libertad de información, accesibilidad

ANOTACIONES

y seguridad. Estos argumentos se han tratado de manera extensa en párrafos anteriores, por lo que estimaré remitirse a ellos.

Por otra parte, ciertamente existen modalidades de licenciamiento por volumen (aunque infortunadamente, el software propietario no satisface los principios básicos). Pero, como Ud. acaba de señalarlo acertadamente en el párrafo inmediatamente anterior de su carta, sólo apuntan a reducir el impacto de un componente que importa no más del 8% del coste total.

Prosigue: "8. Adicionalmente, la alternativa adoptada por el proyecto (i) es claramente más costosa por los altos costes que supone una migración y (ii) pone en riesgo la compatibilidad y posibilidad de interoperabilidad de las plataformas informáticas dentro del Estado, y entre el Estado y el sector privado, dada la centena de versiones que existen de software de código abierto en el mercado."

Analicemos su afirmación en dos partes. Su primer argumento, el de que la migración supone altos costes es en realidad un argumento en favor del proyecto. Porque cuanto más tiempo transcurra la migración a otra tecnología ésta se tornará más onerosa; y al mismo tiempo se irán incrementando los riesgos de seguridad asociados con el software propietario. De esta manera, el uso de sistemas y formatos propietarios va haciendo que el Estado se vuelva cada vez más dependiente de proveedores determinados. Por el contrario, una vez implantada la política de uso de software libre (implantación que, es cierto, implica un coste), la migración de un sistema a otro se hace muy sencilla, ya que todos los datos están almacenados en formatos abiertos. Por otra parte, la migración a un entorno de software abierto no implica más costes que la misma entre entornos distintos de software propietario, con lo que su argumento se invalida totalmente.

El segundo argumento refiere a "dificultades de interoperabilidad de las plataformas informáticas dentro del Estado, y entre el Estado y el sector privado". Esta afirmación implica un cierto desconocimiento de los mecanismos de construcción de software libre, en el que no se maximiza la dependencia del usuario respecto de una plataforma determinada, como sucede habitualmente en el campo del software propietario. Aun cuando existen múltiples distribuciones de software

ANOTACIONES

libre, y numerosos programas susceptibles de ser empleados para una misma función, la interoperabilidad queda garantizada tanto por el empleo de formatos estándar, exigido en el proyecto, como por la posibilidad de construir software interoperable a partir de la disponibilidad del código fuente.

Dice luego que: “9. El software de código abierto en su mayoría no ofrece los niveles de servicio adecuados ni la garantía de fabricantes reconocidos para lograr mayor productividad por parte de los usuarios, lo cual ha motivado que diferentes entidades públicas hayan retrocedido en su decisión de ir por una solución de software de código abierto y se encuentren utilizando software comercial en su lugar.”

Esta observación es infundada. Respecto de la garantía su argumento ha sido rebatido respondiendo el párrafo 4. Respecto de los servicios de soporte, es posible usar software libre sin ellos (así como sucede también con el software propietario) pero quienes los requieran pueden adquirir soporte por separado, tanto de empresas locales cuanto de corporaciones internacionales, también como en el caso de software propietario.

Por otra parte, contribuiría en mucho a nuestro análisis que nos informase acerca de proyectos de software libre *implantados* en entidades públicas, que a la fecha hayan sido abandonados en favor del software propietario. Conocemos un buen número de casos en el sentido inverso, pero carecemos de información respecto de casos en el sentido que Ud. expone.

Continúa observando que: “10. El proyecto desincentiva la creatividad de la industria peruana de software, que factura US$ 40 millones/año, exporta US$ 4 millones (10.º. en el ranking de productos de exportación no tradicional, más que artesanías) y es una fuente de empleo altamente calificado. Con una ley que incentive el uso de software de código abierto, los programadores de software pierden sus derechos de propiedad intelectual y su principal fuente de retribución.”

Está claro por demás que nadie está obligado a comercializar su código como software libre. Tan sólo deberá tener en cuenta que, si no lo hace, no podrá venderle al sector público. Éste, por otra parte, no

ANOTACIONES

constituye el principal mercado para la industria nacional de software. Líneas arriba hemos abordado algunas cuestiones referidas a la influencia del proyecto en la generación de empleo técnico altamente calificado y en mejores condiciones de competitividad, por lo que parece innecesario insistir aquí en el punto.

Lo que sigue en su afirmación es erróneo. Por un lado, ningún autor de software libre pierde sus derechos de propiedad intelectual, a menos que por su expresa voluntad desee colocar su obra en el dominio público. El movimiento del software libre siempre ha sido extremadamente respetuoso de la propiedad intelectual, y ha generado reconocimiento público extenso a los autores. Nombres como el de Richard Stallman, Linus Torvalds, Guido van Rossum, Larry Wall, Miguel de Icaza, Andrew Tridgell, Theo de Raadt, Andrea Arcangeli, Bruce Perens, Darren Reed, Alan Cox, Eric Raymond, y muchos otros, son mundialmente reconocidos por sus contribuciones al desarrollo de software que hoy es utilizado por millones de personas en todo el mundo, en tanto los nombres de los autores materiales de excelentes piezas de software propietario permanecen en el anonimato. Por otra parte, afirmar que las regalías por derechos de autor constituyen la principal fuente de retribución de los programadores peruanos es en todo caso aventurado, en particular porque no se ha aportado ninguna prueba al efecto ni una demostración de cómo el empleo de software libre por el Estado influiría en estas retribuciones.

Prosigue Ud. diciendo que: “11. El software de código abierto, al poder ser distribuido gratuitamente, tampoco permite generar ingresos para sus desarrolladores por medio de la exportación. De esta forma, se debilita el efecto multiplicador de la venta de software a otros países y por lo tanto el crecimiento de esta industria, cuando contrariamente las normas de un gobierno deben estimular la industria local.”

Esta afirmación demuestra nuevamente un desconocimiento total de los mecanismos y el mercado del software libre. Intenta aseverar que el mercado de cesión de derechos no exclusivos de uso a título oneroso (venta de licencias) es el único posible para la industria informática cuando, como Ud. mismo lo ha señalado párrafos arriba, ni siquiera es el más importante. El incentivo que el proyecto presenta al surgimiento de una oferta de profesionales más calificados, en

ANOTACIONES

conjunto con el incremento de experiencia que resultará para los técnicos nacionales el trabajar a gran escala con software libre en el Estado, los colocan en una posición altamente competitiva para brindar sus servicios al extranjero.

Señala luego que "12. En el Foro se discutió sobre la importancia del uso de software de código abierto en la educación, sin comentar el rotundo fracaso de esta iniciativa en un país como México, en donde precisamente los funcionarios del Estado que fundamentaron el proyecto hoy expresan que el software de código abierto no permitió brindar una experiencia de aprendizaje a alumnos en la escuela, no se contó con los niveles de capacitación a nivel nacional para dar soporte adecuado a la plataforma, y el software no contó y no cuenta con los niveles de integración para la plataforma que existen en las escuelas."

Efectivamente, en México se dio marcha atrás con el proyecto Red Escolar. Eso se debió, precisamente a que los impulsores del proyecto mexicano tuvieron al coste de las licencias como principal argumento, en vez de las otras razones estipuladas en nuestro proyecto y que son mucho más esenciales. Debido a este error conceptual, y como consecuencia de la falta de apoyo efectivo por parte de la SEP (Secretaría de Educación Publica) se asumió que para implementar software libre en las escuelas, bastaba con quitarle a éstas el presupuesto para software y en cambio enviarles un CD-ROM con GNU/Linux. Por cierto, esto falló y no podía ser de otro modo, tal como fallan los laboratorios escolares en los que se usa software propietario si no hay presupuesto para implementación y mantenimiento. Es precisamente por eso, por lo que nuestro proyecto de ley no se limita a indicar la mandatoriedad del uso de software libre, sino que reconoce la necesidad y ordena la creación de un plan de migración viable, en el que el Estado encamine ordenadamente la transición técnica para lograr disfrutar de las ventajas del software libre.

Finaliza Ud. con una pregunta retórica: "13. Si el software de código abierto satisface todos lo requerimientos de las entidades del Estado ¿por qué se requiere de una ley para adoptarlo? ¿No debería ser el mercado el que decida libremente cuáles son los productos que le dan más beneficios o valor?".

ANOTACIONES

Estamos de acuerdo que en el sector privado de la economía es el mercado el que debe decidir qué productos usar y allí no sería admisible ninguna intromisión estatal. Pero en el caso del sector público, el razonamiento no es el mismo: como ya establecimos, el Estado almacena, manipula y transforma información que no le pertenece, sino que le ha sido confiada por los ciudadanos que, por imperio de la ley, no tienen más alternativa que hacerlo. Como contraparte a esa imposición legal, el Estado debe extremar las medidas para salvaguardar la integridad, confidencialidad y accesibilidad de esas informaciones. El empleo de software propietario arroja serias dudas sobre el cumplimiento de estos atributos, a falta de evidencia concluyente al respecto y por lo tanto no es apto para ser usado en el sector público.

La necesidad de una ley estriba, por un lado, en la materialización de los principios fundamentales antes enunciados en el campo específico del software. Por otro, en el hecho de que el Estado no es una entidad ideal homogénea, sino que está compuesto de múltiples organismos con diversos grados de autonomía de decisiones. Dado que el software propietario es inapropiado para ser empleado, el hecho de establecer estas reglas en la ley impedirá que la decisión discrecional de cualquier funcionario ponga en riesgo la información que pertenece a los ciudadanos. Y, sobre todo, porque constituye una reafirmación actualizada en relación con los medios de tratamiento y comunicación de información empleados hoy en día sobre el principio republicano de publicidad.

Conforme a este principio universalmente aceptado, el ciudadano tiene derecho a conocer toda información en poder del Estado que no esté amparada en una declaración fundada de secreto conforme a la ley. Ahora bien, el software trata información y es en sí mismo información. Información en formato especial, susceptible de ser interpretada por una máquina para ejecutar acciones, pero sin duda información crucial porque el ciudadano tiene legítimo derecho a saber, por ejemplo, cómo se computa su voto o se calculan sus impuestos. Y para ello, debe poder acceder libremente al código fuente y probar a su satisfacción los programas que se utilizan para el cómputo electoral o para el cálculo de sus impuestos.

ANOTACIONES

6.4.5. Decreto de medidas de impulso de la sociedad del conocimiento en Andalucía

Articulado relacionado con el software libre del decreto de medidas de impulso de la sociedad del conocimiento en Andalucía.

- Artículo 11. Materiales educativos en soporte informático

1. Se dotará a los centros docentes públicos de materiales y programas educativos en soporte informático, basados preferentemente en software libre. En todo caso, recibirán en dicho soporte todo el material educativo que elabore la Administración de la Junta de Andalucía.
2. Asimismo, se incentivará entre el profesorado la producción de programas y materiales curriculares en soporte informático o para su utilización en Internet, especialmente aquellos desarrollos que se realicen mediante software libre.

- Articulo 31. Software libre

1. En las adquisiciones de equipamiento informático destinado a los centros docentes públicos para su uso en actividades educativas, se exigirá que todo el hardware sea compatible con sistemas operativos basados en Software Libre. Los ordenadores tendrán preinstalado todo el Software Libre necesario para el uso específico al que estén destinados.
2. El equipamiento informático que la Administración de la Junta de Andalucía ponga a disposición en los centros de acceso público a Internet utilizará para su funcionamiento productos de Software libre.
3. La Administración de la Junta de Andalucía fomentará la difusión y utilización orientadas al uso personal, doméstico y educativo del Software Libre. A tal fin se establecerá un servicio de asesoramiento a través de Internet para la instalación y uso de este tipo de productos.

- Artículo 49. Objeto

1. Se establecerán ayudas para el desarrollo de proyectos innovadores que faciliten la integración de las tecnologías de la información y las comunicaciones en la formación profesional ocupacional.

ANOTACIONES

2. Estos proyectos irán referidos a alguna de las siguientes modalidades:

a) Elaboración de materiales y contenidos de formación profesional ocupacional para su uso y difusión a través de Internet, especialmente aquellos desarrollos que se realicen mediante software libre.

b) Realización de acciones formativas a través de metodologías innovadoras de tipo semipresencial y a distancia.

ANOTACIONES

7. Ingeniería del software libre

En los capítulos anteriores se ha mostrado por qué el desafío del software libre no es el de un nuevo competidor que bajo las mismas reglas genera software de manera más rápida, barata y de mayor calidad: el software libre se diferencia del software *tradicional* en aspectos más fundamentales, empezando por razones filosóficas y motivaciones, siguiendo por nuevas pautas económicas y de mercado y finalizando con una forma de generar software diferente. La ingeniería del software no puede ser ajena a este suceso y desde hace poco más de un lustro se ha venido profundizando en la investigación de todos estos aspectos. Este capítulo pretende mostrar los estudios más significativos y las evidencias que aportan con el objetivo de ofrecer al lector una visión del estado del arte y de las perspectivas de futuro en lo que hemos venido a denominar la ingeniería del software libre.

7.1. Introducción

Aunque hace ya varias décadas que se desarrolla software libre, sólo desde hace unos pocos años se ha empezado a prestar atención a sus modelos y procesos de desarrollo desde el punto de vista de la ingeniería del software. Igual que no hay un único modelo de desarrollo de software propietario, tampoco lo hay en el mundo del software libre, pero aun así pueden encontrarse interesantes características que comparten gran parte de los proyectos estudiados, y que podrían estar enraizadas en las propiedades de los programas libres.

En 1997, Eric S. Raymond publicó el primer artículo ampliamente difundido, "La catedral y el bazar", donde se trataba de describir algunas características de los modelos de desarrollo de software libre, haciendo especial énfasis en lo que diferencia estos modelos de los de desarrollo propietario. Este artículo se ha convertido desde entonces en uno de los más conocidos (y criticados) del mundo

ANOTACIONES

del software libre, y hasta cierto punto, en la señal de comienzo para el desarrollo de la ingeniería del software libre.

7.2. "La catedral y el bazar"

Raymond establece una analogía entre la forma de construir las catedrales medievales y la manera clásica de producir software. Argumenta que en ambos casos existe una clara distribución de tareas y roles, destacando la existencia de un diseñador que está por encima de todo y que ha de controlar en todo momento el desarrollo de la actividad. Asimismo, la planificación está estrictamente controlada, dando lugar a unos procesos claramente detallados en los que idealmente cada participante en la actividad tiene un rol específico muy delimitado.

Dentro de lo que Raymond toma como el modelo de creación de catedrales no sólo tienen cabida los procesos pesados que podemos encontrar en la industria del software (el modelo en cascada clásico, las diferentes vertientes del Rational Unified Process, etc.), sino que también entran en él proyectos de software libre, como es el caso de GNU y NetBSD. Para Raymond, estos proyectos se encuentran fuertemente centralizados, ya que unas pocas personas son las que realizan el diseño e implementación del software. Las tareas que desempeñan estas personas, así como sus roles, están perfectamente definidos y alguien que quisiera entrar a formar parte del equipo de desarrollo necesitaría que se le asignara una tarea y un rol según las necesidades del proyecto. Por otra parte, las entregas de este tipo de programas se encuentran espaciadas en el tiempo siguiendo una planificación bastante estricta. Esto supone tener pocas entregas del software y ciclos entre las entregas largos y que constan de varias etapas.

El modelo antagónico al de la catedral es el bazar. Según Raymond, algunos de los programas de software libre, en especial el núcleo Linux, se han desarrollado siguiendo un esquema similar al de un bazar oriental. En un bazar no existe una máxima autoridad que controle los procesos que se están desarrollando ni que planifique estrictamente lo que ha de suceder. Por otro lado, los roles de los

ANOTACIONES

participantes pueden cambiar de manera continua (los vendedores pueden pasar a ser clientes) y sin indicación externa.

Pero lo más novedoso de "La catedral y el bazar" es la descripción del proceso que ha hecho de Linux un éxito dentro del mundo del software libre; es una sucesión de *buenas maneras* para aprovechar al máximo las posibilidades que ofrece la disponibilidad de código fuente y la interactividad mediante el uso de sistemas y herramientas telemáticas.

Un proyecto de software libre suele surgir a raíz de una acción puramente personal; la causa se ha de buscar en un desarrollador que vea limitada su capacidad de resolver un problema. El desarrollador debe tener los conocimientos necesarios para, por lo menos, empezar a resolverlo. Una vez que haya conseguido tener algo usable, con algo de funcionalidad, sencillo y, a ser posible, bien diseñado o escrito, lo mejor que puede hacer es compartir esa solución con la comunidad del software libre. Es lo que se denomina la publicación temprana (*release early* en inglés) y que permite llamar la atención de otras personas (generalmente desarrolladores) que tengan el mismo problema y que puedan estar interesados en la solución.

Uno de los principios básicos de este modelo de desarrollo es considerar a los usuarios como codesarrolladores. Ha de tratárseles con mimo, no sólo porque pueden proporcionar publicidad mediante el boca a oreja, sino también por el hecho de que realizarán una de las tareas más costosas que existe en la generación de software: las pruebas. Al contrario que el codesarrollo, que es difícilmente escalable, la depuración y las pruebas tienen la propiedad de ser altamente paralelizables. El usuario será el que tome el software y lo pruebe en su máquina bajo unas condiciones específicas (una arquitectura, unas herramientas, etc.), una tarea que multiplicada por un gran número de arquitecturas y entornos supondría un gran esfuerzo para el equipo de desarrollo.

Si se trata a los usuarios como desarrolladores, puede darse el caso de que alguno de ellos encuentre un error y lo solucione, enviando un parche al desarrollador del proyecto para que el problema esté solucionado en la siguiente versión. También puede

ANOTACIONES

suceder, por ejemplo, que una persona diferente a quien descubra un error sea la que finalmente lo entienda y corrija. En cualquier caso, todas estas circunstancias son muy provechosas para el desarrollo del software, por lo que es muy beneficioso entrar en una dinámica de esta índole.

Lectura complementaria

[Ver bibliografía, apartado 156]

Esta situación se torna más efectiva con entregas frecuentes, ya que la motivación para encontrar, notificar y corregir errores es alta debido a que se supone que va a ser atendido inmediatamente. Además, se consiguen ventajas secundarias, como el hecho de que al integrar frecuentemente –idealmente una o más veces al día– no haga falta una última fase de integración de los módulos que componen el programa. Esto se ha denominado publicación frecuente (en la terminología anglosajona se conoce como *release often*) y posibilita una gran modularidad a la vez que maximiza el efecto propagandístico que tiene el hecho de publicar una nueva versión del software.

Nota

a gestión de nuevas versiones depende lógicamente del tamaño del proyecto, ya que la problemática a la que hay que enfrentarse no es igual cuando el equipo de desarrollo tiene dos componentes que cuando son cientos. Mientras en los proyectos pequeños en general este proceso es más bien informal, la gestión de entregas en los proyectos grandes suele seguir un proceso definido no exento de cierta complejidad. Existe un artículo titulado *Release Management Within Open Source Projects* que describe detalladamente la secuencia que se sigue en el servidor web Apache, el núcleo del sistema operativo Linux y el sistema de versiones Subversion.

Lectura complementaria

[Ver bibliografía, apartado 91]

Para evitar que la publicación frecuente asuste a usuarios que prioricen la estabilidad sobre la rapidez con la que el software evoluciona, algunos proyectos de software libre cuentan con varias ramas de desarrollo en paralelo. El caso más conocido es el del núcleo Linux, que tiene una rama estable –dirigida a aquellas personas que estiman su fiabilidad– y otra inestable –pensada para desarrolladores– con las últimas innovaciones y novedades.

ANOTACIONES

7.3. Liderazgo y toma de decisiones en el bazar

Raymond supone que todo proyecto de software libre ha de contar con un *dictador benevolente*, una especie de líder –que generalmente coincide con el fundador del proyecto– que ha de guiar el proyecto y que se reserva siempre la última palabra en la toma de decisiones. Las habilidades con las que ha de contar esta persona son principalmente las de saber motivar y coordinar un proyecto, entender a los usuarios y codesarrolladores, buscar consensos e integrar a todo aquel que pueda aportar algo al proyecto. Como se puede observar, entre los requisitos más importantes no se ha mencionado el que sea técnicamente competente, aunque nunca está de más.

Con el crecimiento en tamaño y en número de desarrolladores de algunos proyectos de software libre han ido apareciendo nuevas formas de organizar la toma de decisiones. Linux, por ejemplo, cuenta con una estructura jerárquica basada en la delegación de responsabilidades por parte de Linus Torvalds, el *dictador benevolente*. De este modo, vemos cómo hay partes de Linux que cuentan con sus propios *dictadores benevolentes*, aunque éstos vean limitado su *poder* por el hecho de que Linus Torvalds sea el que decida en último término. Este caso es un ejemplo claro de cómo la alta modularidad existente en un proyecto de software libre ha propiciado una forma de organización y toma de decisiones específica.

Nota

Existen voces que dicen que la forma de organizarse dentro de los proyectos de software libre se asemeja a la del equipo quirúrgico propuesta por Harlan Mills (de IBM) a principios de la década de los setenta y popularizada por Brooks en su mítico libro *The Mythical Man-Month*. Aunque se pueden dar casos donde el equipo de desarrollo de alguna aplicación de software libre esté formado por un diseñador-desarrollador principal (el cirujano) y por muchos codesarrolladores que realizan tareas auxiliares (administración de sistemas, mantenimiento, tareas especializadas, documentación, etc.), no existe en ningún caso una separación tan estricta y definida como propone Mills/Brooks. En cualquier caso, como bien apunta Brooks

Lectura complementaria

[Ver bibliografía, apartado 130]

ANOTACIONES

> en el caso del equipo quirúrgico, en el software libre el número de desarrolladores que tienen que comunicarse para crear un sistema grande y complejo es más reducido que el número total de desarrolladores.

En el caso de Fundación Apache nos encontramos con una *meritocracia*, ya que cuenta con un comité de directores formado por personas que han contribuido de manera notable al proyecto. En realidad, no se trata de una rigurosa meritocracia en la que los que más contribuyen son los que gobiernan, ya que el comité de directores es elegido democrática y periódicamente por los miembros de la Fundación Apache (que se encarga de gestionar varios proyectos de software libre, entre ellos Apache, Jakarta, etc.). Para llegar a ser miembro de la Fundación Apache, se deben haber realizado aportaciones de forma continuada e importante a uno o varios proyectos de la fundación. Este sistema también es utilizado en otros proyectos grandes, como es el caso de FreeBSD o GNOME.

Otro caso interesante de organización formal es el *GCC Steering Commitee*. Fue creado en 1998 para evitar que nadie se hiciera con el control del proyecto GCC (GNU Compiler Collection, el sistema de compilación de GNU) y refrendado por la FSF (promotora del proyecto GNU) pocos meses después. En cierto sentido es un comité continuador de la tradición de uno similar que había en el proyecto egcs (que durante un tiempo fue paralelo al proyecto GCC, pero más tarde se unió a éste). Su misión fundamental es asegurarse de que el proyecto GCC respeta la *Declaración de objetivos* (Mission Statement) del proyecto. Los miembros del comité lo son a título personal, y son elegidos por el propio proyecto de forma que representen, con cierta fidelidad, a las diferentes comunidades que colaboran en el desarrollo de GCC (desarrolladores de soporte para diversos lenguajes, desarrolladores relacionados con el kernel, grupos interesados en programación empotrada, etc.).

Lectura complementaria

[Ver bibliografía, apartado 69]

El liderazgo de un proyecto de software libre por parte de una misma persona no tiene por qué ser eterno. Se pueden dar básicamente dos circunstancias por las que un líder de proyecto deje de serlo. La primera de ellas es la falta de interés, tiempo o motivación para seguir adelante. En ese caso, se ha de pasar el testigo a otro desarrollador

ANOTACIONES

que tome el rol de líder del proyecto. Estudios recientes demuestran que en general los proyectos suelen tener un liderazgo que cambia de manos con frecuencia, pudiéndose observar varias *generaciones* de desarrolladores en el tiempo. El segundo caso es más problemático: se trata de una bifurcación. Las licencias de software libre permiten tomar el código, modificarlo y redistribuirlo por cualquier persona sin que haga falta el visto bueno del líder del proyecto. Esto no se suele dar de manera general, salvo en los casos en los que se haga con el fin de evitar deliberadamente precisamente al líder del proyecto (y un posible veto suyo a la aportación). Estamos ciertamente ante una especie de *golpe de Estado*, por otro lado totalmente lícito y legítimo. Es por ello que uno de los objetivos que busca un líder de proyecto al mantener satisfechos a sus codesarrolladores es minimizar la posibilidad de que exista una bifurcación.

7.4. Procesos en el software libre

Aunque el software libre no está necesariamente asociado con un proceso de desarrollo software específico, existe una amplio consenso sobre los procesos más comunes que se utilizan. Esto no quiere decir que no existan proyectos de software libre que hayan sido creados utilizando procesos clásicos como el modelo en cascada. Generalmente, el modelo de desarrollo en proyectos de software libre suele ser más informal, debido a que gran parte del equipo de desarrollo realiza esas tareas de manera voluntaria y sin recompensa económica, al menos directa, a cambio.

La forma como se capturan requisitos en el mundo del software libre depende tanto de la *edad* como del tamaño del proyecto. En las primeras etapas, fundador de proyecto y usuario suelen coincidir en una misma persona. Más adelante, y si el proyecto crece, la captura de requisitos suele tener lugar a través de las listas de correo electrónico y se suele llegar a una clara distinción entre el equipo de desarrollo –o al menos los desarrolladores más activos– y los usuarios. Para proyectos grandes, con muchos usuarios y muchos desarrolladores, la captura de requisitos se hace mediante la misma herramienta que se utiliza para gestionar los errores del proyecto. En este caso, en vez de hablar de errores, se refieren a actividades, aunque el mecanismo utilizado para su gestión es idéntico al de la corrección

ANOTACIONES

de errores (se la clasificará según importancia, dependencia, etc. y se podrá monitorizar si ha sido resuelta o no). El uso de esta herramienta para la planificación es más bien reciente, por lo que se puede observar cómo en el mundo del software libre existe una cierta evolución desde la carencia absoluta a un sistema centralizado de gestión en el ámbito de la ingeniería de actividades, aunque indudablemente éste sea bastante limitado. En cualquier caso, no suele ser común un documento que reúna los requisitos tal y como es normal en el modelo en cascada.

En cuanto al diseño global del sistema, sólo los grandes proyectos suelen tenerlo documentado de manera exhaustiva. En el resto de proyectos, lo más probable es que el o los desarrolladores principales sean los únicos que lo posean –a veces sólo en su mente–, o que vaya fraguándose en versiones posteriores del software. La carencia de un diseño detallado no sólo implica limitaciones en cuanto a la posible reutilización de módulos, sino que también es un gran hándicap a la hora de permitir el acceso a nuevos desarrolladores, ya que éstos se tendrán que enfrentar a un proceso de aprendizaje lento y costoso. El diseño detallado, por su parte, tampoco está muy generalizado. Su ausencia implica que se pierdan muchas posibilidades de reutilización de código.

Lectura complementaria

[Ver bibliografía, apartado 172]

La implementación es la fase en la que se concentra el mayor esfuerzo por parte de los desarrolladores de software libre, entre otras razones porque a ojos de los desarrolladores es manifiestamente la más divertida. Para ello, se suele seguir el paradigma de programación clásico de prueba y error hasta que se consiguen los resultados apetecidos desde el punto de vista subjetivo del programador. Históricamente, es raro el caso en el que se han incluido pruebas unitarias conjuntamente con el código, aun cuando facilitarían la modificación o inclusión de código posterior por parte de otros desarrolladores. En el caso de algunos proyectos grandes, como por ejemplo Mozilla, existen máquinas dedicadas exclusivamente a descargarse los repositorios que contienen el código más reciente para compilarlo en diferentes arquitecturas. Los errores detectados se notifican a una lista de correo de desarrolladores.

Sin embargo, las pruebas automáticas no están muy arraigadas. Por lo general serán los propios usuarios, dentro de la gran variedad de

ANOTACIONES

usos, arquitecturas y combinaciones, los que las realizarán. Esto presenta la ventaja de que se paralelizan a un coste mínimo para el equipo de desarrollo. El problema que plantea este modelo es cómo organizarse para que la realimentación por parte de los usuarios exista y resulte lo más eficiente posible.

En cuanto al mantenimiento del software en el mundo del software libre –refiriéndonos con ello al mantenimiento de versiones antiguas–, es una tarea cuya existencia depende del proyecto. En proyectos donde se requiere una gran estabilidad, como núcleos del sistema operativo, etc., se mantienen versiones antiguas de los mismos, ya que un cambio a una nueva versión puede resultar traumático. Pero, por lo general, en la mayoría de los proyectos de software libre, si se tiene una versión antigua y se encuentra un error, los desarrolladores no suelen ponerse a corregirlo y aconsejan utilizar la versión más moderna con la esperanza de que el error haya desaparecido, ya que el software ha evolucionado.

7.5. Crítica a “La catedral y el bazar”

“La catedral y el bazar” adolece de una falta de sistematicidad y rigor acorde con su naturaleza más bien ensayística y ciertamente poco científica. Las críticas más frecuentes se refieren a que se está contando básicamente una experiencia puntual –el caso de Linux– y que se pretenden generalizar las conclusiones para todos los proyectos de software libre. En este sentido, se puede ver en Sandeep Krishnamurthy. “Cave or community? an empirical examination of 100 mature open source pro-jects”. *First Monday* (vol. 6, núm. 7, mayo, 2002), que la existencia de una comunidad tan amplia como con la que cuenta el núcleo Linux es más bien una excepción.

Lectura complementaria

[Ver bibliografía, apartado 139]

Todavía más críticos se muestran aquellos que piensan que Linux es un ejemplo de desarrollo siguiendo el modelo de las catedrales. Argumentan que parece evidente que existe una cabeza pensante, o al menos una persona que tiene la máxima potestad, y un sistema jerárquico mediante delegación de responsabilidades hasta llegar a los obreros-programadores. Asimismo, existe reparto de tareas, aunque sea de manera implícita. En Nikolai Bezroukov. “A second look at the cathedral and the bazaar”. *First Monday* (vol. 12, núm. 4, di-

Lectura complementaria

[Ver bibliografía, apartado 72]

ANOTACIONES

ciembre, 1998) se va más allá y se sostiene –no sin cierta acritud y arrogancia en la argumentación– que la metáfora del bazar es internamente contradictoria.

Lectura complementaria

[Ver bibliografía, apartados 130 y 128]

Otro de los puntos más criticados de "La catedral y el bazar" es su afirmación de que la ley de Brooks que dice "*agregar desarrolladores a un proyecto de software retrasado lo retrasa aún más*" no es válida en el mundo del software libre. En Paul Jones. (mayo, 2000). *Brook's law and open source: The more the merrier?* se puede leer cómo en realidad lo que pasa es que las condiciones de entorno son diferentes y lo que en un principio aparenta ser un incongruencia con la ley de Brooks, tras un estudio más exhaustivo no deja de ser un espejismo.

7.6. Estudios cuantitativos

El software libre permite ahondar en el estudio cuantitativo del código y del resto de parámetros que intervienen en su generación, dada la accesibilidad a los mismos. Esto permite que áreas de la ingeniería del software tradicional –como la ingeniería del software empírica– se puedan ver potenciadas debido a la existencia de una ingente cantidad de información a la que se puede acceder sin necesidad de una fuerte intromisión en el desarrollo de software libre. Los autores estamos convencidos de que esta visión puede ayudar enormemente en el análisis y en la comprensión de los fenómenos ligados a la generación de software libre (y de software en general) llegándose incluso, entre otras posibilidades, a tener modelos predictivos del software que se realimenten en tiempo real.

La idea que hay detrás es muy simple: dado que tenemos la posibilidad de estudiar la evolución de un número inmenso de proyectos de software libre, hagámoslo. Y es que además del estado actual de un proyecto, toda su evolución en el pasado es pública, por lo que toda esta información debidamente extraída, analizada y empaquetada puede servir como una base de conocimiento que permita evaluar la *salud* de un proyecto, facilitar la toma de decisiones y pronosticar complicaciones actuales y futuras.

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartados 105, 161 y 125]

El primer estudio cuantitativo de cierta importancia en el mundo del software libre data de 1998, aunque fuera publicado a prin-

cipios del 2000. Su finalidad era conocer de manera empírica la participación de los desarrolladores en el mundo del software libre. Para ello se valían del tratamiento estadístico de la asignación de autoría que los autores suelen situar en la cabeza de los ficheros de código fuente. Se demostró que la participación sigue la ley de Pareto: el 80% del código corresponde al 20% más activo de los desarrolladores, mientras que el 80% de desarrolladores restante tiene una aportación de un 20% del total. Muchos estudios posteriores han confirmado y ampliado a otras formas de participación, diferentes a la aportación de código fuente (mensajes a lista de correo, notificación de errores, incluso el número de descargas como se puede ver en Francis Hunt; Paul Johnson. (2002). *On the pareto distribution of sourceforge projects*. Technical report, Centre for Technology Management, Cambridge University Engineering Department, Mill Lane, Cam-bridge CB2 1RX la validez de este resultado.

Nota

El hecho de que aparezcan muchos términos de las ciencias económicas en el estudio de ingeniería del software libre es consecuencia del interés que algunos economistas han mostrado en los últimos tiempos en conocer y entender los procesos que llevan a algunos voluntarios a producir bienes de gran valor sin obtener generalmente beneficio directo a cambio. El artículo más conocido es Rishab A. Ghosh. "Cooking pot markets: an economic model for the trade in free goods and services on theinternet". *First Monday* (vol. 3, núm. 3, marzo, 1998), en el que se introduce la *economía del regalo* en Internet. En "Wikipedia. Pareto" se puede obtener más información sobre el principio de Pareto y su generalización en la distribución que lleva su nombre. Asimismo, son interesantes la curva de Lorenz, que representa de manera gráfica la participación en el proyecto de los desarrolladores, y el coeficiente de Gini, que se calcula a partir de la curva de Lorenz resultando un número a partir del cual se puede ver la desigualdad en el sistema.

Lectura complementaria

[Ver bibliografía, apartados 103, 209, 208 y 207]

La herramienta que se utilizó para la realización de este estudio fue publicada con una licencia libre por los autores, por lo que se ofrece

ANOTACIONES

tanto la posibilidad de reproducir sus resultados como de efectuar nuevos estudios.

Lectura complementaria

[Ver bibliografía, apartado 138]

En un estudio posterior, Koch fue más allá y analizó las interacciones que se llevan a cabo en un proyecto de software libre. Las fuentes de información eran la lista de correo y el repositorio de versiones del proyecto GNOME. Pero el aspecto más interesante del estudio de Koch es el análisis económico. Koch se centra en comprobar la validez de predicción de costes clásicos (puntos de función, modelo de COCOMO, etc.) y muestra los problemas que supone su aplicación, aunque ciertamente admite que los resultados que obtiene –tomados con ciertas precauciones– se ajustan parcialmente a la realidad. Concluye que el software libre necesita de métodos de estudio y modelos propios, ya que los conocidos no se adaptan a su naturaleza. Sin embargo, resulta evidente que la posibilidad de obtener públicamente muchos de los datos relacionados con el desarrollo del software libre permite ser optimistas de cara a la consecución de éstos en un futuro no muy lejano. El de Koch se puede considerar el primer estudio cuantitativo completo, aun cuando ciertamente adolece de una metodología clara y, sobre todo, de unas herramientas *ad hoc* que permitieran comprobar sus resultados y el estudio de otros proyectos.

Lectura complementaria

[Ver bibliografía, apartados 152 y 138]

Mockus y otros presentan en el año 2000 el primer estudio de proyectos de software libre que engloba la descripción completa del proceso de desarrollo y de las estructuras organizativas, incluyendo evidencias tanto cualitativas como cuantitativas. Utilizan para tal fin la historia de cambios del software, así como de los informes de error para cuantificar aspectos de la participación de desarrolladores, el tamaño del núcleo, la autoría de código, la productividad, la densidad de defectos y los intervalos de resolución de errores. En cierto sentido, este estudio no deja de ser un estudio de ingeniería de software clásico, con la salvedad de que la obtención de los datos ha sido realizada íntegramente mediante la inspección semi-automática de los datos que los proyectos ofrecen públicamente en la Red. Al igual que en Stefan Koch; Georg Schneider. "Results from software engineering research into open source development projects using public data. Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informa-tionswirtschaft". En: H.R. Hansen; W.H. Janko (Hrsg.) (2000). *Wirtschaftsuniversität Wien* (núm. 22), con este artículo no se proporcionó ninguna herramienta ni proceso automá-

ANOTACIONES

tico que permitiera ser reutilizada en el futuro por otros equipos de investigación.

En A. David (julio, 2000). *Wheeler. Estimating Linux's size* y A. David (junio, 2001). *Wheeler. More than a gigabuck: Estimating gnu/Linux's* encontramos el análisis cuantitativo de las líneas de código y los lenguajes de programación utilizados en la distribución Red Hat. González Barahona y otros han seguido sus pasos en una serie de artículos dedicados a la distribución Debian (véase por ejemplo Jesús M. González-Barahona; Miguel A. Ortuño Pérez; Pedro de las Heras Quirós; José Centeno González; Vicente Matellán Olivera. (dic, 2001). "Counting potatoes: The size of Debian 2.2.". *Upgrade Magazine* (vol. 6, núm. II, págs. 60-66) y Jesús M. González Barahona; Gregorio Robles; Miguel A. Ortuño Pérez; Luis Rodero Merino; José Centeno González; Vicente Matellán Olivera; Eva M. Castro Barbero; Pedro de las Heras Quirós. "Anatomy of two GNU/Linux distributions". En: Koch. Todos ellos ofrecen una especie de *radiografía* de estas dos populares distribuciones de GNU/Linux realizadas a partir de los datos que aporta una herramienta que cuenta el número de líneas físicas (las líneas de código que no son ni líneas en blanco ni comentarios) de un programa. Aparte del espectacular resultado en líneas de código totales (la última versión estable hasta el momento, Debian 3.0 –conocida como Woody– supera los cien millones de líneas de código), se puede ver la distribución del número de líneas en cada lenguaje de programación. La posibilidad de estudiar la evolución de las diferentes versiones de Debian en el tiempo aporta algunas evidencias interesantes. Cabe mencionar que el tamaño medio de los paquetes permanece prácticamente constante a lo largo de los últimos cinco años, por lo que la tendencia natural de los paquetes a ir creciendo se ve neutralizada por la inclusión de paquetes más pequeños. Por otro lado, se puede percibir cómo la importancia del lenguaje de programación C, todavía predominante, decrece con el tiempo, mientras que lenguajes de guión (Python, PHP y Perl) y Java contabilizan un crecimiento explosivo. Los lenguajes compilados *clásicos* (Pascal, Ada, Modula, etc.) se encuentran en clara recesión. Finalmente, estos artículos incluyen un apartado dedicado a mostrar los resultados obtenidos si se aplica el modelo COCOMO –un modelo de estimación de esfuerzos clásico que data de principios de la década de los ochenta y que se utiliza en proyectos de software propietario– para realizar una estimación de esfuerzo, duración del proyecto y costes.

Lectura complementaria

[Ver bibliografía, apartados 204, 205, 108, 70 y 74]

ANOTACIONES

Aunque son precursores, la mayoría de los estudios presentados en este apartado están bastante limitados a los proyectos que analizan. La metodología que se ha usado se ajusta al proyecto en estudio, es parcialmente manual y en contadas ocasiones la parte automatizada puede ser utilizada de forma general con el resto de proyectos de software libre. Esto supone que el esfuerzo que hay que realizar para estudiar un nuevo proyecto es muy grande, ya que se ha de volver a adaptar el método utilizado y repetir las acciones manuales que se han llevado a cabo.

Lectura complementaria

[Ver bibliografía, apartados 175, 102 y 68]

Por esta razón, los últimos esfuerzos se centran en crear una infraestructura de análisis que integre varias herramientas de manera que se automatice el proceso al máximo. Existen dos motivaciones bastante evidentes para hacerlo así: la primera es que una vez que se ha invertido mucho tiempo y esfuerzo en crear una herramienta para analizar un proyecto –haciendo especial hincapié en que sea genérica–, utilizarla para otros proyectos de software libre implica un esfuerzo mínimo. Por otro lado, el análisis mediante una serie de herramientas que analizan los programas desde diferentes puntos de vista –a veces complementarios, otras veces no– permite obtener una mayor perspectiva del proyecto. En Jesús M. González Barahona; Gregorio Robles. *Libre software engineering web site* se pueden seguir con mayor detenimiento estas iniciativas.

7.7. Trabajo futuro

Después de presentar la breve pero intensa historia de la ingeniería del software aplicada al software libre, podemos afirmar que la ingeniería del software libre se encuentra todavía dando sus primeros pasos. Quedan varios aspectos de gran importancia pendientes de ser estudiados y examinados minuciosamente para dar con un modelo que explique, al menos en parte, cómo se genera software libre. Las cuestiones que se han de afrontar en un futuro próximo son: la clasificación de los proyectos de software libre, la creación de una metodología basada en lo posible en elementos de análisis automáticos y la utilización de los conocimientos adquiridos para la realización de modelos que permitan entender el desarrollo de software libre, a la vez que facilite la toma de decisiones a partir a la experiencia adquirida.

ANOTACIONES

Un aspecto que tampoco se debe dejar de lado, y que ya se está abordando en la actualidad, es la evaluación de la validez de los métodos de ingeniería clásica en el campo del software libre en todas las intensificaciones de la ingeniería del software. Así, por ejemplo, las leyes de la evolución del software postuladas por Lehman a principios de la década de los setenta y actualizadas y aumentadas en los ochenta y noventa parece que no se cumplen incondicionalmente en el desarrollo de algunos proyectos de software libre.

Lectura complementaria

[Ver bibliografía, apartados 144 y 178]

En la actualidad, una de las deficiencias más importantes es la falta de una clasificación estricta del software libre que permita catalogar los proyectos en diferentes categorías. A día de hoy, los criterios de clasificación son demasiado burdos, incluyendo en el mismo saco a proyectos cuyas características organizativas, técnicas o de cualquier otra índole son muy dispares. El argumento de que Linux, con una amplia comunidad de desarrolladores y un gran número, tiene una naturaleza diferente y no se comporta de idéntica manera que un proyecto mucho más limitado en número de desarrolladores y usuarios, es muy acertado. En cualquier caso, una clasificación más exhaustiva permitiría reutilizar la experiencia adquirida en proyectos similares (léase con características parecidas), facilitaría las predicciones, posibilitaría el pronóstico de riesgos, etc.

El segundo aspecto importante al que debe enfrentarse la ingeniería del software libre, muy ligado al punto anterior y a las tendencias actuales, es la creación de una metodología y de herramientas que la soporten. Una metodología clara y concisa permitirá estudiar todos los proyectos con el mismo rasero, averiguar su estado actual, conocer su evolución y, por supuesto, clasificarlos. Las herramientas son esenciales a la hora de abordar este problema, ya que, una vez creadas, permiten tener al alcance de la mano el análisis de miles de proyectos con un mínimo esfuerzo adicional. Uno de los objetivos de la ingeniería del software libre es que a partir de un número limitado de parámetros que indiquen dónde encontrar en la Red información sobre el proyecto (la dirección del repositorio de versiones del software, el lugar donde se almacenan los archivos de las listas de correo-e, la localización del sistema de gestión de errores y una mínima encuesta) se pueda realizar un estudio exhaustivo del proyecto. A los gestores del proyecto sólo les separaría un botón de un análisis completo, una especie de *análisis clínico* que permita juzgar la *salud* del

proyecto que a la vez incluya una indicación sobre los aspectos que necesitan ser mejorados.

Lectura complementaria

[Ver bibliografía, apartado 64]

Una vez que se consigan métodos, clasificación y modelos, las posibilidades que ofrece la simulación y, siendo más concretos, los agentes inteligentes, pueden ser enormes. Habida cuenta de que partimos de un sistema cuya complejidad es notoria, resulta interesante la creación de modelos dinámicos en los que los diferentes entes que participan en la generación de software puedan ser modelados. Evidentemente, cuanto más sepamos de los diferentes elementos, más ajustado a la realidad será el modelo. Aunque se conocen varias propuestas de simulaciones para el software libre, éstas son bastante simples e incompletas. En cierta medida, esto se debe a que todavía existe una gran laguna de conocimiento con respecto a las interacciones que tienen lugar en la generación de software libre. Si se consigue empaquetar y procesar convenientemente la información de los proyectos a través de toda su historia, los agentes pueden ser cruciales para conocer cómo será la evolución en el futuro. Aunque existen muchas propuestas sobre cómo se ha de enfocar este problema, una de las más avanzadas por ahora se puede encontrar en I. Antoniades; I. Samoladas; I. Stamelos; G.L. Bleris. “Dynamical simulation models of the open source development process”. En: Koch.

7.8. Resumen

En resumen, en este capítulo hemos podido ver cómo la ingeniería del software libre es un campo joven y todavía por explorar. Sus primeros pasos se deben a escritos ensayísticos en los que se proponía –no sin cierta falta de rigor científico– un modelo de desarrollo más eficiente, pero paulatinamente se ha ido progresando en el estudio sistemático del software libre desde una óptica de ingeniería. En la actualidad, tras varios años de informes y estudios de proyectos libres cuantitativos y cualitativos completos, se está realizando un enorme esfuerzo en la consecución de una infraestructura global que permita la clasificación, el análisis y la modelización de los proyectos en un espacio de tiempo limitado y de manera parcialmente automática. Cuando el análisis de los proyectos de software libre deje de ser tan costoso en tiempo y en esfuerzo como lo es hoy en día, es muy probable que se abra una nueva etapa en la ingeniería del software

ANOTACIONES

libre, en la que entrarán en escena otro tipo de técnicas, cuyo propósito principal se sitúe en torno a la predicción de la evolución del software y en el pronóstico de posibles complicaciones.

8. Entornos y tecnologías de desarrollo

Aquí se hace una introducción a las tecnologías y entornos utilizados habitualmente para el desarrollo de software libre, y al impacto que ejercen sobre la gestión y evolución de los proyectos.

8.1. Caracterización de entornos, herramientas y sistemas

Antes de explicar herramientas concretas, vamos a hacer definir las características y propiedades generales que estas herramientas tienen, en función del trabajo a realizar y el modo de organizarse los desarrolladores.

En primer lugar, aunque no es determinante, el entorno y las herramientas de desarrollo, así como la máquina virtual objetivo deberían ser a su vez *libres*. Ello no fue así en un principio. En efecto, el proyecto GNU, con el objetivo de reemplazar Unix, tuvo que desarrollarse sobre y para Unix propietarios, hasta la aparición de los Linux y BSD libres. Hoy día, especialmente cuando el software libre se desarrolla como parte de un modelo de negocio, se tiende a que la máquina objetivo pueda también ser un sistema propietario, a menudo por medio de máquinas virtuales interpuestas (Java, Python, PHP, etc.). En todo caso, el entorno y la máquina virtual han de estar suficientemente difundidas y ser lo bastante económicas como para poder reunir suficientes codesarrolladores que dispongan de esas herramientas.

En segundo lugar, también para atraer el mayor número de codesarrolladores, las herramientas deben ser *sencillas*, ampliamente *conocidas* y capaces de funcionar en máquinas *económicas*. Posiblemente por las razones anteriores, el mundo del software libre es relativamente conservador en lenguajes, herramientas y entornos.

En tercer lugar, el modelo de desarrollo es eminentemente distribuido, con muchos colaboradores potenciales repartidos por todo el

ANOTACIONES

mundo. Esto requiere herramientas de colaboración, generalmente asíncronas, que permitan además que el trabajo avance con facilidad, independientemente de la cantidad y ritmo de trabajo producido por cada colaborador, sin retrasar a nadie.

Finalmente, es conveniente proporcionar a los desarrolladores aquellos recursos de los que carecen, como máquinas de arquitecturas diversas, donde puedan compilar y probar sus programas.

8.2. Lenguajes y herramientas asociadas

Lectura complementaria

[Ver bibliografía, apartado 182]

La mayoría del software libre está realizado en lenguaje C, no sólo porque C es el lenguaje natural de toda variante de Unix (plataforma habitual del software libre), sino por la amplia difusión del mismo, tanto en las mentes como en las máquinas (**gcc** es un compilador estándar instalado por defecto en casi todas las distribuciones). Precisamente por estas razones y por su eficiencia lo recomienda Stallman en los proyectos de GNU. Otros lenguajes que se le acercan bastante son C++, también soportado por **gcc** por defecto, y Java, con cierta semejanza y bastante popular por permitir el desarrollo para máquinas virtuales disponibles en gran variedad de plataformas. Generalmente no se tienen en cuenta razones de ingeniería software: en SourceForge (ver apartado 8.9.1), por cada 170 proyectos en C hay uno en Ada, lenguaje supuestamente más apropiado para desarrollar programas de calidad. Del mismo modo, el inglés es la *lingua franca* entre los desarrolladores de software libre, a pesar de que el esperanto es un idioma mucho más fácil de aprender y con una estructura más lógica. También son populares aquellos lenguajes interpretados diseñados para prototipado rápido de aplicaciones normales y servicios web, como Perl, Python y PHP.

Lectura complementaria

[Ver bibliografía, apartados 30, 81, 35 y 7]

Así como C es el lenguaje estándar, **make** es la herramienta estándar de construcción de programas, dados sus fuentes. El programador libre usará normalmente la versión de GNU mucho más que la de incompatible de BSD. Con ellas puede especificar árboles de dependencias entre ficheros y reglas para generar los ficheros dependientes a partir de aquellos de los que dependen. Así, se puede especificar que un fichero objeto `x.o` depende de los ficheros fuente `x.c` y `x.h` y que para construirlo hay que ejecutar **gcc -c x.c**. O que el ejecutable de nuestro pro-

grama depende de una colección de objetos y se monta de una determinada manera. Cuando se modifica un código fuente y luego se ejecuta **make**, se recompilarán solamente los módulos afectados y se volverá a montar el objeto final. Es una herramienta de muy bajo nivel, ya que, por ejemplo, es incapaz de averiguar por sí mismo cuándo se debe recompilar un módulo en C, a pesar de que podría averiguarlo examinando las cadenas de *includes*. También es muy potente, porque puede combinar todas las herramientas disponibles de transformación de ficheros para construir objetivos muy complejos de un proyecto multilenguaje. Pero es muy complicado y muy dependiente de entornos tipo Unix. No obstante, alternativas supuestamente mejores, como Jam o Ant son muy poco usadas.

Dada la heterogeneidad de sistemas existentes incluso en el mundo Unix, también se usan herramientas dedicadas a ayudarnos a que nuestros programas sean portátiles. Las herramientas de GNU Autoconf, Automake y Libtool facilitan estas tareas en entornos C y Unix. Dada la heterogeneidad de idiomas, juegos de caracteres y entornos culturales, el programador de C (y de muchos otros lenguajes), recurre a Gettext y a las facilidades de internacionalización de la biblioteca estándar de C para realizar programas fácilmente localizables dinámicamente.

Lectura complementaria

[Ver bibliografía, apartados 10, 27, 29, 26 y 28]

Así, cuando nosotros recibimos un paquete fuente, lo más probable es que esté escrito en C, empaquetado con **tar**, comprimido con **gzip**, hecho portátil con **autoconf** y herramientas asociadas, y construible e instalable con **make**. Su instalación se llevará a cabo por medio de un proceso muy similar al siguiente:

```
tar xzvf paquete-1.3.5.tar.gz
cd paquete-1.3.5
./configure
make
make install
```

8.3. Mecanismos básicos de colaboración

El software libre es un fenómeno que es posible debido a la colaboración de comunidades distribuidas y que, por tanto, necesitan he-

ANOTACIONES

rramientas que hagan efectiva esa colaboración. Aunque durante mucho tiempo se utilizó correo postal de cintas magnéticas, el desarrollo ágil del software libre empezó cuando fue posible comunicarse rápidamente con muchas personas y distribuirles las fuentes de los programas o responder con comentarios y parches. Por conveniencia, mejor que enviar código, en los mensajes podría difundirse información sobre un sitio donde recogerlo. De hecho, muy al principio de los años setenta, el correo electrónico fue una extensión del protocolo de transferencia de ficheros de ARPANET.

En el mundo Unix, a mediados de los setenta, se desarrolla **uucp**, el protocolo de transferencia de ficheros de Unix, apto para comunicar máquinas por líneas conmutadas, además de dedicadas, sobre el que se montó correo electrónico, y en 1979 el primer enlace USENET sobre UUCP. Las *news* (noticias) de USENET, un sistema de foro temático estructurado jerárquicamente y distribuido por inundación a los sitios suscritos a una jerarquía, jugaron un papel fundamental en el desarrollo libre de software, enviándose programas completos en fuente a los grupos de la jerarquía `comp.sources`.

Lectura complementaria

[Ver bibliografía, apartado 38]

En paralelo se desarrollaron las listas de correo, entre las que cabe mencionar los gestores de listas de BITNET (1981). Hoy día existe la tendencia a preferir las listas de correo a los grupos de noticias tipo USENET. La razón principal ha sido el abuso con fines comerciales y la intrusión de gente *despistada*, que introduce ruido en las discusiones. Además, en las listas de correo hay más control y pueden llegar a más gente. Los destinatarios deben suscribirse y cualquier dirección de correo es válida, aunque no haya acceso directo a Internet. El administrador de la lista puede elegir conocer quién se suscribe o dar de baja a alguien. Puede restringir las contribuciones a los miembros o puede elegir moderar los artículos, antes de que aparezcan, aunque no son prácticas populares. La administración de las listas se ha hecho tradicionalmente por correo electrónico, por medio de mensajes especiales con contraseña. Ello permite que el administrador no tenga acceso permanente a Internet, aunque cada vez es un fenómeno más raro, de modo que el gestor de listas de correo más popular hoy día, Mailman, no puede ser administrado por correo electrónico y tiene que hacerse necesariamente vía web. Las listas de correo juegan un papel crucial en el software libre, llegando en muchos casos a ser el único método de contribución.

ANOTACIONES

Hoy día, con la popularidad de la web, muchos foros son puros foros web o *weblogs*, sin otra interfaz que la que se ofrece a través del navegador. Éstos pueden ser genéricos, como los populares SlashDot o BarraPunto, donde se anuncia nuevo software libre o se discuten noticias relacionadas, o especializados en un programa concreto, que normalmente están integrados en sitios de colaboración con diversas herramientas adicionales (ver apartado 8.9). También hay interfaces web a grupos de noticias y listas tradicionales.

Lectura complementaria

[Ver bibliografía, apartados 48 y 11]

Asimismo, se ha hecho popular un mecanismo de colaboración basado en wikis, sobre todo cuando se trata de elaborar un documento conjunto, que puede ser la especificación de un programa, módulo o sistema. Hablaremos de él en el apartado 8.5.2

Finalmente, cabe mencionar los mecanismos de interacción donde los desarrolladores conversan en tiempo real. Para software libre no suele ser un mecanismo práctico porque con desarrolladores distribuidos por todo el mundo no es fácil encontrar una hora apropiada para todos. No obstante, hay varios proyectos que hacen uso de herramientas de charla textual, ya sea regularmente o en *congresos virtuales* con fechas acotadas. La herramienta más usada es el IRC (*Internet Relay Chat*), que normalmente comunica gente por medio de *canales* temáticos establecidos por medio de una serie de servidores colaboradores. No es común que se utilicen herramientas multimedia (sonido, imagen), probablemente porque puede requerir conexiones de calidad no disponibles para todos, además de problemas de disponibilidad de software libre y la dificultad de registrar y editar los resultados de las conversaciones, con el propósito de documentar.

Lectura complementaria

[Ver bibliografía, apartado 131]

8.4. Gestión de fuentes

A todo proyecto de desarrollo de programas le conviene tener archivada la historia del mismo. Por ejemplo, porque alguna modificación pudo producir un error oculto que se descubre tardíamente y hay que recuperar el original, al menos para analizar el problema. Si el proyecto lo desarrollan varias personas, es necesario también registrar al autor de cada cambio, con las razones de éste debidamente argumentadas. Si del proyecto van haciéndose entregas por versiones, es nece-

ANOTACIONES

sario saber exactamente qué versiones de cada módulo forman dichas entregas. Muchas veces, un proyecto mantiene una versión estable y otra experimental; ambas hay que mantenerlas, corregir sus errores y transferir errores corregidos de una versión a la otra. Todo ello puede hacerse guardando y etiquetando convenientemente todas y cada una de las versiones de los ficheros, lo que generalmente se ha considerado como un coste excesivo, aunque con los discos actuales empiece a no ser tan cierto. Lo que normalmente hace un *sistema de control de fuentes*, también llamado *sistema de gestión de versiones*, es registrar la historia de los ficheros como un conjunto de diferencias sobre un patrón, normalmente el más reciente, por eficiencia, etiquetando además cada diferencia con los metadatos necesarios.

Pero también queremos que un sistema de éstos sirva para que muchos programadores colaboren efectivamente, sin pisarse el trabajo, pero sin detener el avance de cada uno. Debe permitirnos, pues, que haya varios programadores trabajando conjuntamente, pero con un cierto control. Este control puede ser optimista o pesimista. Con un control pesimista, un programador puede reservarse unos ficheros para una mejora por un tiempo, durante el cual nadie puede tocar estos ficheros. Esto es muy seguro, pero bloqueará a otros programadores y el proyecto puede retrasarse, sobre todo si el que bloqueó los ficheros está ocupado en otras cosas, o incluso se olvidó de ellos. Permitir a otros avanzar es más dinámico, pero más peligroso, ya que puede haber modificaciones incompatibles. Un sistema optimista deja avanzar, pero nos avisa cuando ha habido conflictos y nos proporciona herramientas para resolverlos.

8.4.1. CVS

Lectura complementaria

[Ver bibliografía, apartados 17, 93 y 77]

CVS (*Concurrent Version Sytem*) es un sistema de gestión de fuentes optimista diseñado a finales de los ochenta, que es el usado por la inmensa mayoría en los proyectos libres. En el entorno de software libre se trabaja normalmente con un repositorio central al que se accede según un sistema cliente/servidor. El administrador del sitio decide quiénes tienen acceso al repositorio o a qué partes del repositorio, aunque normalmente, una vez que un desarrollador ha sido admitido en el círculo de confianza, tiene acceso a todos los ficheros. Del mismo modo, puede permitirse un acceso anónimo, sólo en lectura, a todo el mundo.

El colaborador anónimo

El *CVS anónimo* es una herramienta fundamental para realizar el concepto de *entregar pronto y frecuentemente* defendido por Eric Raymond. Cualquier usuario ansioso de probar la última versión de un programa la puede extraer del CVS, descubrir errores y reportarlos, incluso en forma de parches con la corrección. Y puede examinar la historia de todo el desarrollo.

Veamos un poco de mecánica. Un usuario avanzado quiere obtener la última versión del módulo `mod`, con un repositorio accesible anónimamente en `progs.org`. directorio `/var/lib/cvs` y protocolo `pserver`. La primera vez declarará su intención de acceder:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs
login
```

Nos pide una contraseña, que será la del usuario anónimo (normalmente, retorno de carro), que se registrará en un fichero local (realmente esta operación no es necesaria para acceso anónimo, pero el programa se queja si no existe el fichero con la contraseña). Seguidamente, lo importante es obtener la primera copia del módulo:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs co
mod
```

Esto creará un directorio `mod` con todos los ficheros y directorios del módulo y ciertos metadatos (contenidos en subdirectorios llamados `CVS`), que nos permitirán, entre otras cosas, no tener que repetir la información ya dada. Nuestro usuario avanzado se introduce en el directorio creado, genera el paquete, y prueba:

```
cd mod
./configure
make
make install
...
```

Cuando quiere una nueva versión, simplemente actualiza su copia dentro de `mod`.

ANOTACIONES

```
cd mod
cvs update
./configure
make
make install
...
```

Si resulta que descubre un error, puede corregirlo en el sitio y luego mandar un parche por correo electrónico al mantenedor del programa (individual o lista de correo):

```
cvs diff -ubB | mail -s "Mis parches" mod-
maint@progs.org
```

El desarrollador normal

El desarrollador normal tendrá una cuenta en el servidor. Puede usar el mismo mecanismo y protocolo que el usuario anónimo, cambiando `anonymous` por el nombre de su cuenta.

Nota

Por seguridad, para cuentas con derecho a escritura, se suele usar ssh, que proporciona un canal autenticado y cifrado.

Una vez tiene una copia de trabajo del módulo, puede hacer las modificaciones necesarias y, cuando considere que se han estabilizado, *comprometer* los cambios en el repositorio. Por ejemplo, si modifica los ficheros `parte.h` y `parte.c`, los comprometerá así:

```
cvs ci parte.h parte.c
```

Antes de terminar la operación, el CVS nos pedirá una explicación de lo que se hizo, que se adjuntará al historial de ambos ficheros. Asimismo, incrementará el *número de revisión* de cada fichero en una unidad. Este número identifica cada momento importante en la historia de un fichero y puede usarse para recuperar cada uno de esos momentos.

¿Cuándo debe un desarrollador realizar una operación de compromiso? Ésta es una cuestión metodológica que deben acordar los miembros de un proyecto, pero parece obvio que no deben comprometerse cambios que no compilen. Pero es preferible que además pasen una batería de pruebas mínima. En muchos proyectos es necesario, asimismo, contar con la aprobación de un supervisor de proyecto o subproyecto que examine la modificación.

Durante el desarrollo de la modificación, alguien puede haber alterado otros ficheros o incluso los mismos. Por ello, conviene que el desarrollador haga con relativa frecuencia una operación de actualizar su copia (**cvs update**). Si han modificado otros ficheros, puede haber cambiado el entorno y fallar pruebas que antes pasaban. Si han modificado los mismos ficheros, puede ser en lugares o rutinas que no hemos tocado o en código que sí que hemos modificado. En el primer caso no hay conflicto (al menos aparente) y la operación de modificación *mezcla* nuestra versión con la del repositorio, generándose ficheros combinados, con todas las modificaciones. De lo contrario, hay conflicto, en cuyo caso hay que ponerse de acuerdo con el desarrollador que hizo los otros cambios y acordar una versión final.

Para identificar mejor cada componente de un proyecto, es conveniente que lleve asociada directamente información de revisión. CVS puede marcar los códigos fuentes y los objetos automáticamente, siempre y cuando observemos cierta disciplina. Por ejemplo, si en un comentario del código fuente escribimos la palabra clave `$Id$`, cada vez que el fichero se comprometa en el repositorio, dicha palabra se sustituirá por una cadena de identificación, donde podemos ver: nombre de fichero, número de revisión, fecha y hora del compromiso, y autor del mismo:

```
$Id: parte.c,v 1.7 2003/07/11 08:20:47 joaquin
Exp $
```

Si esa palabra clave la incluimos dentro de una cadena de caracteres del programa, al compilarlo la cadena aparecerá en el objeto y el ejecutable, con lo que es posible identificarlo con una herramienta (**ident**).

ANOTACIONES

ANOTACIONES

El administrador

Obviamente, los administradores son los que tienen y deben llevar la parte más complicada del mantenimiento del repositorio. Por ejemplo, tiene que dar de alta el proyecto, conceder permisos a los desarrolladores y coordinarlos, etiquetar versiones entregadas, etc.

Es práctica común que todo proyecto tenga una versión estable y otra experimental. Para ello se crean ramas. Mientras que los que se dedican al mantenimiento corrigen errores de la rama estable, los nuevos desarrollos se hacen sobre la rama experimental. Cuando ésta se estabiliza, hay que pasarla a estable, no sin antes aplicar las correcciones hechas sobre la rama estable anterior. Esta operación se llama *mezclar*, es delicada y está soportada en CVS, aunque quizá de forma demasiado primitiva. Esta idea puede extenderse al concepto de ramas experimentales evolucionando en distintas direcciones, llegando o no a buen puerto, que en todo caso, a menos que sean vías muertas, habrá que integrar total o parcialmente en el producto estable, con mezclas apropiadas.

Un derecho que nos da el software libre es modificar un programa para uso privado. Aunque es deseable contribuir con toda mejora al acervo comunitario, muchas veces las modificaciones que queremos hacer son demasiado específicas y no interesan al público en general. Pero sí que nos interesa incorporar la evolución del programa original. Esto puede realizarse con un tipo especial de ramificación y mezcla (*ramas de vendedor*).

El administrador también puede facilitar la coordinación del equipo por medio de mecanismos automáticos, como hacer que se produzcan mensajes de correo electrónico cuando ocurran ciertas cosas, por ejemplo, los compromisos. O forzar a que se realicen ciertas acciones automáticas antes de realizar un compromiso, como comprobaciones automáticas de estilo, compilaciones o pruebas.

8.4.2. Otros sistemas de gestión de fuentes

CVS, a pesar de ser el sistema de control de versiones más ampliamente usado, presenta notables inconvenientes, que no han sido suficientes para sustituirlo a pesar de la necesidad manifiesta de hacerlo así.

- No soporta renombramientos o cambios de directorio de ficheros, ni tampoco metadatos (propietarios, permisos, etc.) ni enlaces simbólicos.

- Al ser una evolución de un sistema de control de versiones de ficheros individuales, no soporta naturalmente el control de versiones de grupos completos.

- No soporta conjuntos de cambios coherentes. En efecto, para añadir una característica o corregir un error, puede ser preciso cambiar varios ficheros. Estos cambios deberían ser atómicos.

- En CVS es bastante complicado el uso de ramas y mezclas. En efecto, si creamos una rama experimental de un proyecto y deseamos incluir las correcciones hechas a la rama estable, es preciso conocer en detalle qué correcciones se han hecho ya, para no repetirlas.

- Aunque la mayoría del trabajo puede hacerse desconectado del servidor, en CVS necesitamos conexión con el mismo para cualquier operación que no sea trabajar con nuestra copia local de ficheros.

- CVS no genera, sin la ayuda de herramientas aparte, el fichero `changelog`, que muestra la historia global de cambios de un proyecto.

- CVS no permite, con facilidad, utilizar herramientas arbitrarias para ver las diferencias entre versiones o para mezclar ficheros que han dado lugar a conflicto. Esto puede ser especialmente importante para ficheros binarios, que, aunque están soportados por CVS, tienen un soporte primitivo. Cuestión que se solucionaría si pudiéramos ver las distintas versiones de los ficheros en el sistema de ficheros.

- CVS carece de una interfaz gráfica, aunque existen varios visores vía web que facilitan la navegación por las distintas versiones, su comparación gráfica, etc.

Sin embargo, existen otros sistemas libres que solucionan varios de estos problemas. Por ejemplo, el veterano Aegis, liberado por primera vez en 1991 por Peter Miller, y casi tan veterano como CVS, pero

Lectura complementaria

[Ver bibliografía, apartados 151, 3 y 123]

poco popular, quizá porque apareció después, porque impone más controles –como la obligación a pasar las pruebas o a pasar por el revisor de código– o porque es más difícil de instalar un sistema de colaboración distribuida. Otros sistemas recientes no han alcanzado el grado necesario de madurez o son complejos, poco portátiles o con una cultura subyacente más propia del software propietario. Vesta, por ejemplo, se integra con la generación de código en una herramienta más monolítica (sistema de gestión de configuraciones).

Lectura complementaria

[Ver bibliografía, apartados 52, 8 y 13]

Podemos destacar el ya designado sucesor de CVS: Subversion, basado en extensiones de HTTP (WebDAV), o Arch, más ambicioso, que soporta repositorios distribuidos. Ésta es una característica muy interesante, que permite a cualquiera tener un repositorio con una rama privada o pública que luego puede mezclar o no con la oficial. Así funciona BitKeeper, elegido por Linus Torvalds para mantener el núcleo de Linux, decisión muy criticada porque el producto tiene una licencia que permite a los proyectos libres obtenerlo gratuitamente, siempre que todas las operaciones de compromiso de cambios con sus metadatos se registren en un servidor público designado por los propietarios y accesible a todo el mundo, y siempre que el licenciatario no esté desarrollando otro sistema de control de fuentes que pueda competir con él. Los argumentos para ello son: no es posible un modelo de negocio de software libre basado en servicios para este segmento del mercado, y no es posible desarrollar un producto libre de tanta calidad sin copiarle las ideas, lo que se facilita dando acceso al producto.



Sugerencia

Pueden verse las operaciones realizadas sobre los códigos fuentes de Linux con BitKeeper en http://linux.bkbits.net.

8.5. Documentación

En el mundo del software libre, apenas se usan procesadores de texto WYSIWYG y otras herramientas ofimáticas que tanto éxito tienen

ANOTACIONES

en otros entornos, a pesar de existir ya herramientas libres, como OpenOffice. Ello es debido a varios factores importantes:

- Es conveniente mantener la documentación bajo control de fuentes, y los sistemas de control de fuentes, como CVS, aunque admiten formatos binarios, prefieren formatos textuales transparentes, editables con un editor de texto normal y procesables con herramientas desarrolladas para programas, que nos permitan ver fácilmente las diferencias entre versiones, generar y aplicar parches basados en esas diferencias y realizar operaciones de mezcla.

  Nota

  En Unix, las herramientas que hacen estas operaciones más comunes son **diffs**, **diff3**, **patch** y **merge**.

- Ciertas licencias de documentación libre, especialmente la GFDL (ver apéndice A), exigen formatos transparentes, sobre todo por facilitar el trabajo a los que realicen documentos derivados.

- Las herramientas WYSIWYG (*what you see is what you get*) generalmente no contienen más información que la estricta de visualización, haciendo muy difícil, si no imposible, el procesamiento automático, como identificar autores o título, y la conversión a otros formatos. Incluso aunque permitan conversión de formatos, ésta suele hacerse de forma interactiva, siendo muchas veces imposible automatizarla (con **make**, por ejemplo).

- Generalmente, las herramientas ofimáticas generan unos formatos de ficheros muy voluminosos, asunto muy desagradable para los desarrolladores o los repositorios.

Por todo ello el programador libre, acostumbrado también a programar y compilar, prefiere formatos de documentación transparentes, en muchos casos simple texto puro, en muchos otros formatos de documentación procesables.

Los formatos procesables utilizados no son muchos. Tradicionalmente, en el mundo Unix los programas se han documentado en los formatos esperados por la familia de procesadores **roff**, con versión libre de Norman Walsh. No obstante, esta práctica se ha ido aban-

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartados 31, 53 y 136]

donando, excepto para las páginas de manual tradicionales, ya que es casi obligado preparar páginas de manual para las herramientas más básicas del sistema. Debido a que muchas páginas de manual han crecido excesivamente, de modo que difícilmente se les puede llamar *páginas*, fue necesario elaborar un formato alternativo hipertextual, que permitiera seguir documentos estructurados con índices y referencias cruzadas. El proyecto GNU diseñó el formato *texinfo* y lo convirtió en su estándar. Este formato permite obtener documentos navegables con la herramienta **info** o dentro del editor **emacs**, y a su vez obtener documentos impresos de calidad con ayuda del procesador de textos TeX, de Donald Knuth.

El formato texinfo puede traducirse a HTML multipágina si se desea, y mucha gente prefiere ver la información con un navegador web, pero se pierde la capacidad de búsqueda de palabras en un documento. Ésta es una de las consecuencias indeseables de la popularidad de HTML, ya que los navegadores no implementan el concepto de documento multipágina, a pesar de que existen elementos `link` que permiten enlazar las partes.

Lectura complementaria

[Ver bibliografía, apartados 142 y 109]

Existe una imperiosa demanda de que los documentos complejos se puedan ver como páginas web multipágina fácilmente navegables. Hay gente que escribe documentación en LaTeX, también aplicación de TeX, muy popular entre los científicos, más expresivo que Texinfo, y convertible a HTML multipágina con ciertas herramientas, siempre que se guarde cierta disciplina. En efecto, las aplicaciones de TeX son conjuntos de macros que combinan operadores tipográficos de muy bajo nivel para convertirlos en lenguajes abstractos que trabajan con conceptos de alto nivel (autor, título, resumen, capítulo, apartado, etc.). Si sólo se utilizan las macros básicas, la conversión es sencilla. Pero como nadie impide usar operadores de bajo nivel y, además, existen cantidades enormes de paquetes de macros fuera de la capacidad de mantenimiento de los mantenedores de los conversores, es difícil conseguir que las conversiones salgan bien.

8.5.1. Docbook

Lectura complementaria

[Ver bibliografía, apartados 63, 201 y 202]

El problema radica en que no existe separación entre contenido y presentación, ni en las aplicaciones de TeX ni en las de **nroff**, ya que la abstracción se construye por capas. Esta separación la tienen las

ANOTACIONES

aplicaciones de SGML y XML, donde la presentación se especifica con *hojas de estilo* completamente separadas. Muy pronto empezaron a utilizarse aplicaciones muy sencillas de SGML, como `linuxdoc` y `debiandoc`, pero debido a su escasa capacidad expresiva, se optó por ir a `DocBook`.

`DocBook` es una aplicación de SGML originalmente desarrollada para documentación técnica informática, y hoy con una variante XML. DocBook es hoy el estándar de formato de documentación libre para muchos proyectos (Linux Documentation Project, Kde, GNOME, Mandrake, etc.) y un objetivo a alcanzar en otros (Linux, *BSD, Debian, etc.).

Sin embargo. DocBook es un lenguaje complejo, plagado de etiquetas, por lo que es conveniente disponer de herramientas de ayuda a la edición, aún escasas y elementales, siendo la más popular el modo `psgml` de **emacs**. También es pesado de procesar y los procesadores libres todavía generan una calidad de documentos poco atractiva.

8.5.2. Wikis

Mucha gente encuentra demasiado complicado escribir documentación con lenguajes tan complejos como DocBook y mecanismos de colaboración como CVS. Por ello se ha hecho muy popular un nuevo mecanismo de colaboración para la elaboración de documentos en línea vía web llamado wiki, inventado por Ward Cunningham[ward:wiki], puesto por primera vez en servicio en 1995, y hoy ampliamente utilizado en muy diversas variantes para elaborar documentos muy dinámicos, no destinados a ser impresos, y muchas veces con una vida corta (por ejemplo, organización de una conferencia).

Al contrario que DocBook, un wiki tiene un lenguaje de marcas muy simple y conciso, que recuerda la presentación final, sin ser exactamente como ella. Por ejemplo, los párrafos se separan por una línea en blanco, los elementos de listas empiezan por un guión si no se numeran y por un cero si se numeran, o las celdas de tablas se separan por barras verticales y horizontales.

Tampoco existe el concepto de documento completo, sino que un wiki es más bien un conjunto de pequeños documentos enlazados

ANOTACIONES

que se van creando a medida que es necesario explicar un nuevo concepto o tema. La creación de los documentos es casi automática, ya que la herramienta de edición muestra muy claramente que hemos introducido un concepto (a través de un `NombreWiki`, casi siempre dos palabras juntas con la primera letra en mayúsculas). Casi ningún wiki admite hiperenlaces dentro de la misma página.

Al contrario que en CVS, cualquiera puede escribir en un wiki, aunque se aconseja que se identifique al autor con un registro previo. Cuando se visita un wiki, veremos que todas las páginas tienen un botón para permitir su edición. Si se pulsa, el navegador nos mostrará en un formulario el código fuente del documento, que podremos cambiar. No es una edición WYSIWYG, lo que disuade al que quiera fastidiar, pero es tan sencillo que cualquier interesado puede modificar documentos con muy pequeño esfuerzo.

Los wikis llevan su propio control de versiones de documentos, de modo que generalmente están accesibles todas sus versiones con indicación de quién las hizo y cuándo. También se pueden comparar con facilidad. Suelen llevar mecanismos de búsqueda, al menos por nombre de página y por palabra contenida.

Normalmente, el autor original de una página querrá enterarse de las modificaciones que se le hacen. Para ello puede suscribirse a los cambios, recibiendo notificaciones de los mismos por correo electrónico. A veces el que ve un documento no se atreve a cambiar nada, pero puede hacer un comentario. Normalmente, toda página wiki tiene asociado un foro de comentarios que se pegan al final del documento y que, bien el autor original, bien alguien que asuma el rol de editor, puede emplearlos para reformar el texto original, posiblemente moviendo frases de los comentarios a los sitios oportunos.



Sugerencia

La mejor forma de entender un wiki es accediendo a uno y experimentando en una página destinada a ello, denominada SandBox. Puede acceder a cualquier de ellas, como por ejemplo, http://twistedmatrix.com/users/jh.twistd/moin/moin.cgi/WikiSandBox

8.6. Gestión de errores

Uno de los puntos fuertes del modelo de desarrollo libre es que la comunidad contribuya con informes de errores, y que ésta sienta que sus informes o soluciones son tenidos en cuenta. Por ello, es necesario un mecanismo sencillo de informar de errores, de modo que los desarrolladores reciban información suficiente, de forma sistemática, y con todos los detalles necesarios, ya sea aportados por el colaborador, como la explicación de lo que pasa, nivel de importancia y posible solución, ya por algún mecanismo automático que determine, por ejemplo, la versión del programa y del entorno en que funciona. Asimismo, es necesario que los errores se guarden en una base de datos que pueda ser consultada, para ver si un error ya ha sido reportado, si ha sido corregido, su nivel de importancia, etc.

Existen varios de estos sistemas, de filosofía diferente. Unos son vía web, otros, vía correo electrónico, por medio de algún programa intermediario. Todos tienen interfaz web para consulta. Unos permiten informes anónimos, otros requieren identificación (una dirección de correo válida), para evitar el ruido. Aunque los procedimientos vía web parecen los más sencillos, con ellos no es posible obtener fácilmente información automática del entorno del error. Por ejemplo, el sistema de Debian proporciona programas como **reportbug**, que tras preguntar el nombre del paquete sobre el que queremos informar, consulta al servidor qué errores ya hay reportados sobre el mismo. Si ninguno de ellos se refiere a nuestro problema, nos pide una descripción del mismo, un nivel de importancia (crítico, grave, serio, importante, no se puede regenerar a partir de los códigos fuentes, normal, menor o sugerencia) y etiquetas sobre la categoría (por ejemplo, seguridad). Tras ello, si confirmamos la petición, automáticamente averigua la versión del paquete y de aquellos de los que depende, así como la versión y arquitectura del núcleo. Obviamente, la dirección de correo electrónico la sabe, por lo que un informe similar al siguiente es enviado al sitio correcto:

```
Package: w3m-ssl
Version: 0.2.1-4
Severity: important
After reloading a page containing complex ta-
bles several dozen times, w3m had
```

ANOTACIONES

```
used all physical memory and thrashing commen-
ced. This is an Alpha machine.
-- System Information
Debian Release: testing/unstable
Kernel Version: Linux romana 2.2.19 #1 Fri Jun
1 18:20:08 PDT 2001 alpha unknown
Versions of the packages w3m-ssl depends on:
ii libc6.1 2.2.3-7 GNU C Library: Shared libra-
ries and Timezone data
ii libgc5 5.0.alpha4-8 Conservative garbage co-
llector for C
ii libgpmg1 1.19.3-6 General Purpose Mouse Li-
brary [libc6]
ii libncurses5 5.2.20010318-3 Shared libraries
for terminal handling
ii libssl0.9.6 0.9.6a-3 SSL shared libraries
ii w3m 0.2.1-2 WWW browsable pager with tables/
frames support
```

Este mensaje genera un número de error que nos es devuelto, enviado al mantenedor y guardado en la base de datos. Cuando se resuelva el error, recibiremos también una notificación. Cada error tiene asignada una dirección de correo electrónico que se puede utilizar para proporcionar información adicional, por ejemplo. La base de datos de errores la podremos consultar vía http://bugs.debian.org en cualquier momento.

A veces los sistemas de seguimiento de errores tienen mecanismos para asignar la corrección a alguien y ponerle un plazo. En este caso, el sistema se acerca a un sistema de gestión y seguimiento de flujo de trabajos, a veces integrados con el sistema de gestión de fuentes.

8.7. Sistemas de gestión de flujo de trabajos

En software libre no se emplean mecanismos muy sofisticados de gestión de los trabajos que tiene que hacer cada desarrollador. Después de todo, muchos colaboradores son voluntarios y no se les pue-

ANOTACIONES

Lectura complementaria

[Ver bibliografía, apartados 1 y 71]

de poner un esquema muy rígido. No obstante, sí que se pueden definir tareas y esperar a que alguien se dé de alta en el sistema y las asuma, declarando un plazo para ello. Se tenga o no, control sobre lo que pueden hacer determinadas personas, siempre es conveniente tener controladas las tareas que hay que hacer, qué dependencias tienen, su nivel de importancia y quiénes están trabajando en ellas. El sistema más sofisticado que se está utilizando para esto es Issuezilla, una evolución propietaria del sistema de gestión de errores libre vía web BugZilla.

8.8. Soporte para otras arquitecturas

El soporte mínimo que podemos desear para hacer nuestro programa portátil a otros entornos distintos del que poseemos es el acceso a *granjas de compilación*, que permiten compilar nuestro programa en otras arquitecturas y sistemas operativos. Por ejemplo, SourceForge (apartado 8.9.1) ofrecía, cuando se escribió esto, entornos GNU Linux Debian para Intel x86, DEC Alpha, PowerPC y SPARC, además de entornos Solaris y Mac OS/X.

Seguidamente, también nos gustaría poder probar nuestros programas en esos entornos. A medida que pedimos más servicios de máquinas ajenas ofrecidas como servicio público, más difícil será obtenerlos. El mismo servicio de compilación puede ser problemático, ya que nosotros podemos querer desarrollar para lenguajes diversos, con bibliotecas diversas. La granja de compilación puede proporcionar un número limitado de ellos. Pero si queremos probar un programa ordinario, las dificultades aumentan exponencialmente, no sólo porque es muy difícil disponer de los programas requeridos, sino por razones de seguridad, que pueden hacer muy complicado administrar estos sistemas. No obstante, existen unos pocos servicios de *granja de servidores*, con instalaciones estándar de arquitecturas diversas, que pueden permitirnos probar algunas cosas. Por ejemplo, aparte de lo que se pueda hacer en la granja de servidores de SourceForge, podemos encontrar hasta granjas de PDA.

Lectura complementaria

[Ver bibliografía, apartado 44]

Las granjas públicas descritas anteriormente son un servicio de uso manual. El desarrollador invitado copia sus ficheros en una de esas máquinas, los compila y prueba el resultado. Probablemente debe

Lectura complementaria

[Ver bibliografía, apartado 40]

ANOTACIONES

hacerlo de vez en cuando, en el momento previo a la liberación de una versión importante del programa. Puede ser mucho más interesante que las compilaciones y la ejecución de las pruebas de regresión se hagan sistemáticamente, de forma automática, por ejemplo todas las noches, si ha habido cambio en los códigos fuentes. Así funcionan algunos proyectos importantes, que proporcionan una infraestructura propia para desarrolladores externos, que suele llamarse *Tinderbox* (*yesquero*). Es el caso de Mozilla, financiado por Netscape, cuyo *Tinderbox* presenta una interfaz web a los resultados de la compilación y pruebas de componentes de su navegador en todas las arquitecturas donde funciona. Esta interfaz está íntimamente relacionada con el CVS y muestra esos resultados para distintos estados (entre compromisos), identificando al responsable de los fallos y facilitando el avance, soslayando el problema hasta que se solucione. También usan yesqueros los proyectos OpenOffice y FreeBSD, por lo menos.

8.9. Sitios de soporte al desarrollo

Los sitios de soporte al desarrollo proporcionan, de manera más o menos integrada, todos los servicios descritos anteriormente, junto con algunos adicionales que permiten la búsqueda de proyectos por categorías y su clasificación según parámetros sencillos de actividad. Ello libera al desarrollador de montarse y administrarse toda una infraestructura de colaboración y concentrarse en su proyecto.

8.9.1. SourceForge

Lectura complementaria

[Ver bibliografía, apartado 51]

Uno de los primeros en establecerse este tipo de servicios y el más popular es SourceForge, gestionado por OSDN (Open Software Development Network, subsidiaria de VA Software), que albergaba en junio del 2003 más de 60.000 proyectos. Está estructurado en torno a un conjunto de programas del mismo nombre, que hasta la versión 2 fue software libre.

SourceForge, como prototipo de estos sitios, ofrece una interfaz web o portal global de entrada (http://sourceforge.net/) y un subportal por proyecto (http://proyecto.sourceforge.net). En la interfaz global

ANOTACIONES

podemos ver noticias, anuncios, enlaces y una invitación a hacerse miembro o entrar, si ya se es. Para colaborar en el sitio, conviene hacerse miembro, siendo imprescindible hacerlo si se quiere crear un proyecto nuevo o participar en uno ya existente. Para ser espectador no es necesario, y como tal podemos ver cuáles son los proyectos que experimentan un desarrollo más activo o son descargados con más frecuencia; podemos buscar proyectos por categorías o dando una palabra de su descripción, y se nos mostrarán ordenados por su nivel de actividad. Dentro de cada proyecto, podemos ver su descripción, estado (alfa, beta, producción), descriptores (lenguaje, sistema operativo, temática, tipo de usuarios, idioma, licencia), errores y temas pendientes o subsanados, detalles de su actividad en el tiempo, o descargarlo. También podemos participar en foros o informar de errores, incluso de forma anónima, lo cual no es muy conveniente (por ejemplo, no nos pueden responder).

Cualquier usuario autenticado puede solicitar dar de alta un proyecto, que es admitido por los administradores del sitio si cumple las políticas del mismo, y que en el caso de SourceForge son bastante liberales. Una vez autorizado, el creador puede dar de alta a otros usuarios registrados como administradores adicionales o como desarrolladores, con acceso a la modificación de fuentes. Tras la autorización, no hay muchos más controles sobre el proyecto, lo que ocasiona la existencia de muchos proyectos muertos. Esto no confunde demasiado a los usuarios porque los proyectos con baja o nula actividad apenas son visibles, ya que se muestran ordenados por ella en las búsquedas. Estos proyectos corren el riesgo de ser eliminados por los propietarios del sitio. Los servicios que SourceForge proporciona a un proyecto, y que podríamos esperar de cualquier servicio similar, son:

- Albergue para las páginas web del portal del proyecto, en la dirección *proyecto.sourceforge.net,* donde se muestra el mismo al público. Estas páginas pueden ser estáticas o dinámicas (con CGI o PHP), en cuyo caso pueden hacer uso de una base de datos (MySQL). Se introducen directamente a través de órdenes de copia remota y pueden manipularse por medio de sesiones interactivas de terminal remoto (ssh).

- Opcionalmente, un servidor virtual que responda a direcciones de un dominio obtenido aparte, como *www.proyecto.org* o *cvs.proyecto.org*.

ANOTACIONES

- Tantos foros web y/o listas de correo como sean necesarios, según criterio de un administrador.

- Un servicio de noticias, donde los administradores anuncian novedades sobre el proyecto.

- Rastreadores (*trackers*), para informe y seguimiento de errores, peticiones de soporte, peticiones de mejoras o integración de parches. Los administradores dan una prioridad al asunto y asignan su solución a un desarrollador.

- Gestores de tareas. Similar a un rastreador, permite definir subproyectos con una serie de tareas. Estas tareas además de una prioridad tienen un plazo. Los desarrolladores a los que se les asignan pueden manifestar de vez en cuando un porcentaje de realización de la tarea.

- Un CVS con derechos iniciales de acceso para todos los desarrolladores.

- Servicio de subida y bajada de paquetes de software. Utilizándolo se tiene un registro de las versiones introducidas y se posibilita que los interesados reciban un aviso cuando esto suceda. Además, la subida implica la creación de varias réplicas en todo el mundo, lo que facilita la distribución.

- Servicio de publicación de documentos en HTML. Cualquiera puede registrarlos, pero sólo después de la aprobación por un administrador serán visibles.

- Copia de seguridad para recuperación de desastres, como rotura de disco, no de errores de usuario, como puede ser borrar un fichero accidentalmente.

Un usuario autenticado tiene una página personal donde se reúne toda la información de su interés, como proyectos a los que está asociado, temas o tareas que tiene pendientes, así como foros y ficheros que ha declarado que quiere vigilar. Además, para que no tenga que estar pendiente de su página personal, un usuario recibirá por correo electrónico notificaciones respecto de lo que quiere vigilar.

ANOTACIONES

8.9.2. Herederos de SourceForge

En el 2001 VA Software estuvo a punto de quebrar, en plena crisis de las *puntocom*. Entonces anunció una nueva versión de su software SourceForge con licencia no libre, en un intento de procurarse una fuente de ingresos, vendiéndolo a empresas para sus desarrollos internos. Asimismo, eliminó mecanismos que permitían volcar un proyecto para llevarlo a otro sitio. Ambos hechos fueron vistos como una amenaza por la que los miles de proyectos alojados en SourceForge quedarían atrapados en manos de una sola empresa, que utilizaría la plataforma para demostrar software no libre. Ante eso y la posibilidad de que el sitio cerrara se desarrollaron hijos de la versión libre y se abrieron portales basados en ellas, entre los que destacan Savannah, dedicado al proyecto GNU y a otros programas con licencia tipo copyleft, o BerliOS, concebido como punto de encuentro entre desarrolladores de software libre y empresas. Sin embargo, esto es sólo un paso con vistas a desarrollar una plataforma distribuida y replicada, donde nadie tenga control absoluto de los proyectos.

Lectura complementaria

[Ver bibliografía, apartados 47, 12 y 79]

8.9.3. Otros sitios y programas

Naturalmente se han desarrollado y siguen desarrollándose sistemas de colaboración y algunas empresas hacen negocio del mantenimiento y servicio de esos sitios. Por ejemplo, el proyecto Tigris, que sólo mantiene proyectos de ingeniería de software libre, usa un portal de colaboración (SourceCast) mantenido por una empresa de servicios (CollabNet), que también mantiene sitios de proyectos individuales, como OpenOffice. Nuevos sitios vienen adoptando nuevo software libre, como GForce, utilizado por el proyecto Debian. Puede verse una comparación exhaustiva de muchos sitios en Haggen So. *Comparison of free/open source hosting (fosphost) sites available for hosting projects externally from project owners*.

Lectura complementaria

[Ver bibliografía, apartados 54, 25, 6 y 181]

ANOTACIONES

9. Estudio de casos

Este capítulo está dedicado íntegramente a estudiar más a fondo los proyectos de software libre clásicos más interesantes en cuanto a impacto en el mundo del software libre, resultados obtenidos, modelo de gestión, evolución histórica, etc. Por supuesto, el número de proyectos que se van a presentar es muy pequeño en comparación con el número de proyectos libres totales –que se calcula que asciende a unos 50 mil–, por lo que este capítulo no se podría considerar completo, ni se podrá considerar nunca completo. Aun así, esperamos que tras su lectura el lector pueda tener al menos una percepción de cómo se ha llevado a la práctica la teoría que se ha ido presentando a lo largo de este libro.

Los proyectos escogidos abarcan desde aplicaciones de bajo nivel –ésas que interactúan más bien con el sistema físico del ordenador en vez de con el usuario– hasta entornos de trabajo para el usuario final. Hemos incluido también proyectos de software libre que, en un principio, no son íntegramente de desarrollo. Tal es el caso de las distribuciones, cuyo trabajo es más bien de integración, ya que se dedican principalmente a tomar un conjunto amplio, pero limitado, de aplicaciones independientes y a crear un sistema donde todo interactúe de manera efectiva, incluyendo también facilidades para instalar, actualizar y borrar nuevas aplicaciones según lo desee el usuario.

Los proyectos de más bajo nivel que vamos a ver van a ser Linux, el núcleo del sistema operativo más popular a día de hoy, y FreeBSD, que conjunta un núcleo de la familia BSD con una serie de aplicaciones y utilidades realizados por terceros proyectos al más puro estilo de las distribuciones. Los entornos de trabajo de usuario final escogidos para su estudio serán KDE y GNOME, ciertamente los más extendidos y populares. Los servidores –uno de los puntos fuertes de los sistemas libres– estarán representados en este capítulo por Apache, líder en el mercado de los servidores WWW. Asimismo, contaremos con la presentación de Mozilla, uno de los clientes WWW (en realidad, mucho más que eso) con los que contamos en el mundo del

ANOTACIONES

software libre. El último proyecto que se va a ver en este capítulo es OpenOffice.org, una suite de oficina libre.

Para terminar, hemos creído conveniente mostrar con un poco de mayor profundidad dos de las distribuciones más populares: Red Hat Linux y Debian GNU/Linux y realizar una comparación en cuanto a tamaño con otros sistemas ampliamente utilizados como pueden ser los de Microsoft Windows o Solaris.

Lectura complementaria

[Ver bibliografía, apartado 74]

Al término de las presentaciones de los diferentes casos de estudio, mostraremos un cuadro con las características más notables de cada aplicación o proyecto. Uno de los parámetros que puede sorprender más al lector son los resultados de estimación de coste, de duración y de número medio de desarrolladores. Estos resultados los hemos obtenido por medios tradicionales en la ingeniería del software, en especial el modelo COCOMO. El modelo COCOMO toma como medida de entrada el número de líneas de código fuente y genera estimaciones de coste total, tiempo de desarrollo y esfuerzo dedicado. COCOMO es un modelo pensado para procesos de generación de software *clásicos* (desarrollo en cascada o en V) y para proyectos de tamaño medio o grande, por lo que las cifras que nos ofrece en nuestro caso han de ser tomadas con mucho cuidado. En cualquier caso, los resultados nos pueden dar una idea del orden de magnitud en el que nos movemos, con información sobre los esfuerzos óptimos necesarios si se hubiera utilizado un modelo de desarrollo propietario.

Lectura complementaria

[Ver bibliografía, apartado 212]

En general, lo que más asombra de los resultados de COCOMO es su estimación de costes. En dicha estimación se tienen en cuenta dos factores: el salario medio de un desarrollador y el factor de *overhead*. En el cálculo de la estimación de costes, se ha tomado el salario medio para un programador de sistemas a tiempo completo de acuerdo con la encuesta del año 2000, con Computer World (2000). *Salary survey 2000*.. El *overhead* es el sobrecoste que toda empresa ha de asumir para que el producto salga a la calle con independencia del salario de los programadores. En este apartado se incluyen desde el salario para las secretarias y el equipo de marketing hasta los costes de las fotocopias, luz, equipos hardware, etc. En resumen, el coste calculado por COCOMO es el coste total que le supondría a una empresa crear un software del tamaño especificado y no se ha

ANOTACIONES

de ver simplemente como el dinero que percibirían los programadores por realizar el software. Una vez dicho esto, los cálculos de costes dejan de parecer tan abultados.

9.1. Linux

El kernel (núcleo en castellano) Linux es, sin duda, la aplicación estrella del software libre, hasta el punto de que, aun siendo una pequeña parte del sistema, ha dado nombre a todo él. Es más, se podría incluso afirmar que el propio software libre se confunde en multitud de ocasiones con Linux, lo que no deja de ser un gran desacierto, ya que existe software libre que corre sobre sistemas que no se basan en Linux (de hecho, una de las grandes metas del movimiento y de muchos proyectos de software libre es que las aplicaciones puedan ejecutarse en multitud de entornos). Por otro lado, también existen aplicaciones que funcionan en Linux y que no son software libre (Acrobat Reader, el lector de documentos PDF, también cuenta con su versión para Linux).

Nota

Para evitar la asociación del software libre únicamente con sistemas Linux existen varios proyectos que se dedican a integrar y distribuir aplicaciones libres que se corren en sistemas Windows. Uno de estos proyectos, probablemente el más conocido y completo, es GNUWin, que distribuye en un CD autoarrancable más de un centenar de aplicaciones libres para sistemas Win32. La mayoría de estas aplicaciones también se encuentran disponibles en las distribuciones GNU/Linux comunes, por lo que son una buena herramienta para ir preparando el cambio de sistemas Windows a GNU/Linux de manera más sosegada.

9.1.1. Historia de Linux

La historia de Linux es una de las historias más conocidas dentro del mundo del software libre, seguramente porque se asemeja más a un cuento que a la historia de un programa de ordenador. En 1991, un

ANOTACIONES

estudiante finés de nombre Linus Torvalds decidió que quería aprender a utilizar el modo protegido 386 en una máquina que su limitado bolsillo le había permitido adquirir. Por entonces –y aún hoy en día– en los cursos de sistemas operativos universitarios existía un núcleo de sistema operativo gestado con fines académicos llamado Minix. A la cabeza del grupo de desarrollo de Minix –basado en los sistemas UNIX tradicionales– se encontraba uno de los profesores de universidad más prestigiosos, Andrew Tanenbaum. Minix era un sistema limitado, pero bastante capaz y bien diseñado que contaba a su alrededor con una gran comunidad académica y de ingeniería.

Minix tenía una licencia de libre distribución y se podía utilizar sin problema para fines académicos, pero tenía el gran hándicap de que personas ajenas a la Universidad de Amsterdam no podían integrar mejoras en él, sino que lo debían hacer de manera independiente, generalmente a través de parches. Esto suponía que en la práctica existiera una versión de Minix oficial que todo el mundo usaba y luego una larga serie de parches que había que aplicar posteriormente para obtener funcionalidades adicionales.

A mediados de 1991, Linus –el todavía anónimo estudiante finés– mandó un mensaje al grupo de noticias de Minix anunciando que iba a empezar desde cero un núcleo de sistema operativo basado en Minix, pero sin incluir código del mismo. Para entonces, Linus –aun cuando no dijera explícitamente que lo fuera a publicar con una licencia libre– apuntó que el sistema que iba a crear no tendría las *barreras* que presentaba Minix, por lo que –sin saberlo, y probablemente sin quererlo– estaba dando el primer paso para quedarse con la comunidad que entonces se aglutinaba alrededor de Minix.

Lectura complementaria

[Ver bibliografía, apartado 106]

La versión 0.02 que data de octubre de 1991, aunque muy limitada, ya podía ejecutar terminales bash y el compilador GCC. En los meses siguientes el número de aportaciones externas fue creciendo hasta el punto de que ya en marzo de 1992 Linus publicaba la versión 0.95, que fue ampliamente reconocida como casi estable. El camino hacia la versión 1.0 –que suele asociarse con la estabilidad– fue, sin embargo, todavía largo: en diciembre de 1993 se publicaría por ejemplo la versión 0.99pl14 (que viene a ser como la decimocuarta versión corregida de la versión 0.99) y finalmente en marzo de 1994

Linux 1.0 vio la luz. Ya para entonces, Linux se publicaba bajo las condiciones de la licencia GPL, según el propio Torvalds una de las mejores decisiones que ha tomado, ya que ayudó sobremanera a la distribución y popularización de su núcleo. En Michael W. Godfrey; Qiang Tu. "Evolution in open source software: A case study". *Proceedings of the 2000 International Conference on Software Maintainance* (agosto, 2000) se puede encontrar un análisis exhaustivo de la evolución de las diferentes versiones del kernel de Linux en cuanto a tamaño y modularidad.

Nota

Para los anales de la historia del software libre queda también el debate que tuvo lugar a finales de enero de 1992 en el grupo de noticias de Minix entre Andrew Tanenbaum y Linus Torvalds. Tanenbaum, probablemente ya un poco molesto por el éxito de Torvalds con su *juguete*, atacó de manera un poco desproporcionada a Linux y a Linus. El punto esencial es que Linux era un sistema monolítico (el núcleo es una pieza que integra todos los manejadores, etc.) y no microkernel (el núcleo tiene un diseño modular, lo que permite que sea más pequeño y que se puedan cargar módulos bajo demanda). Se puede seguir la discusión original tal y como ocurrió en el grupo de noticias en Andrew Tanenbaum; Linus Torvalds. (1999). *The tanenbaum-torvalds debate*.

Lectura complementaria

[Ver bibliografía, apartado 193]

9.1.2. El modo de trabajo de Linux

La forma de trabajar de Torvalds no era muy común en aquellos tiempos. El desarrollo se basaba principalmente en una lista de correo. En la lista de correo no sólo se discutía, sino que también se desarrollaba. Y es que a Torvalds le gustaba mucho que toda la vida del proyecto se viera reflejada en la lista, por lo que pedía que los parches se enviaran a la misma. En contra de lo que uno pudiera esperar –que se enviara el parche como adjunto– Linus prefería que se mandara el código en el cuerpo del mensaje para que él y los demás pudieran comentarlo. En cualquier caso, aun cuando muchos opinaran y enviaran correcciones o nuevas funcionalidades, lo que era in-

ANOTACIONES

discutible es que la última palabra, quien decidía lo que entraba en Linux, correspondía a Linus Torvalds.

Nota

La consolidación de Linus Torvalds como *dictador benevolente* ha dado pie a un amplio anecdotario dentro del proyecto. Así, se comenta que si una idea gusta, se ha de implementar. Si no gusta, también se ha de implementar. El corolario, por tanto, es que las buenas ideas no sirven para nada (sin código, por supuesto). Por otro lado, si no gusta la implementación, hay que insistir. Conocido es el caso de Gooch, para quien el santo Job era un aprendiz. Gooch realizó hasta 146 parches paralelos hasta que Linus decidió finalmente integrarlo en la rama oficial del núcleo.

Otra de las ideas innovadoras de Torvalds fue el desarrollo paralelo de dos ramas del núcleo: la estable (cuyo segundo número de versión suele ser par, como por ejemplo 2.4.18) y la inestable (impar, como 2.5.12). Como no podía ser de otra forma, es Torvalds quien decide qué entra en uno y en otro sitio (muchas de las decisiones más polémicas las podemos ubicar precisamente aquí). En cualquier caso, Linux no tiene una planificación de entregas con fechas fijas: estará listo cuando esté listo, mientras tanto habrá que esperar. El lector seguro que habrá asumido a estas alturas que la decisión de si está listo o no lo está corresponde, como no podría ser de otra forma, únicamente a Linus.

El método de desarrollo utilizado en Linux resulta, tal como se ha demostrado, muy eficaz en cuanto a resultados: Linux goza de una gran estabilidad y existe generalmente un intervalo ínfimo de corrección de errores (a veces del orden de los minutos), ya que cuenta con miles de desarrolladores. En esta situación, cuando existe un error, la probabilidad de que uno de ellos lo encuentre es muy alta y, en caso de que no lo pueda resolver el descubridor, alguno ya dará con la solución de manera rápida. En definitiva, podemos ver cómo Linux cuenta con miles de personas/mes dedicadas a su desarrollo, lo que indudablemente hace que su éxito no sea nada sorprendente.

ANOTACIONES

Se debe mencionar que esta forma de trabajar es, por contra, muy cara en cuanto a recursos se refiere. No es inusual que existan muchas propuestas mutuamente excluyentes para una nueva funcionalidad o que se reciban una docena de parches para el mismo error. En la gran mayoría de los casos, solamente una de ellas será incluida en el núcleo finalmente, por lo que se puede considerar que el resto del tiempo y esfuerzo dedicado por los desarrolladores ha sido en balde. El modelo de desarrollo de Linux es, por tanto, un modelo que funciona muy bien en Linux, pero que ciertamente no todos los proyectos se pueden permitir.

9.1.3. Estado actual de Linux

En la actualidad Linux se encuentra en el camino de su versión 2.6, aunque este dato todavía no se sepa a ciencia cierta, ya que hay voces que solicitan que la siguiente versión tenga un número de versión 3.0, debido principalmente a la inclusión de NUMA (acceso a memoria no uniforme, utilizada de manera común en multiprocesadores). Sin duda, ésta no será la única mejora, sino que se mejorarán e incluirán muchas cosas más, como sistemas de ficheros, mejoras para la comunicación en redes inalámbricas y arquitecturas de sonido (ALSA), entre otras. Remitimos al lector interesado al siguiente artículo Joseph Pranevich (2003). "The wonderful world of Linux 2.6, actualizado con cierta frecuencia.

Lectura complementaria

[Ver bibliografía, apartado 165]

En cuanto al modelo de desarrollo de Linux, éste ha sufrido algunos cambios en los últimos años. Aunque la lista de correo de desarrollo sigue siendo el *alma* del proyecto, el código ya no ha de pasar necesariamente por ella. A ello ha contribuido en gran manera BitKeeper, un sistema de control de versiones propietario desarrollado por la compañía BitMover siguiendo las estrictas recomendaciones de Linus Torvalds. El uso de una herramienta no libre ha generado una gran polémica, en la que se ha podido constatar otra vez la posición pragmática de Linus (para él, y para muchos más, el sistema libre de control de versiones CVS está muy anticuado).

A modo de resumen, en el siguiente cuadro se ofrece una radiografía del proyecto Linux. Podemos ver cómo Linux ha superado hace poco la barrera de los tres millones de líneas de código, pudiéndose encuadrar entre los proyectos muy grandes. De hecho, Linux es, junto

ANOTACIONES

con Mozilla y OpenOffice.org, uno de los proyectos más grandes dentro del panorama del software libre. En cuanto a la estimación de tiempo de ejecución y número medio de desarrolladores que lo harían óptimo, podemos observar cómo el primero es ciertamente menor que los diez años de historia con los que cuenta Linux. Esto, por otro lado, se compensa con creces teniendo en cuenta el segundo, ya que el número medio de desarrolladores a tiempo completo es superior del que dispone Linux.

Nota

La estimación de costes que nos ofrece COCOMO es de 131 millones de dólares americanos, una cifra que, para ponerlo en un contexto de las cifras que se suelen manejar más asiduamente, supone el doble de lo que los grandes clubes de fútbol suelen pagar por una gran estrella.

Tabla 9-1. Estado actual de Linux

Página web	http://www.kernel.org
Inicio del proyecto	Primer mensaje en news.comp.os.minix: Agosto 1991
Licencia	GNU GPL
Versión actual de Linux	2.4.21 (versión estable del 16.6.2003)
Líneas de código fuente	3.238.908
Estimación de coste (según COCOMO básico)	$ 131.000.000
Estimación de tiempo de ejecución (según COCOMO básico)	7,31 años (87,71 meses)
Estimación de número medio de desarrolladores (según COCOMO básico)	132
Número aproximado de desarrolladores	Se cuentan por miles (aunque en los créditos aparecen sólo centenares [tuomi:linux-credit-files:02])
Herramientas de ayuda al desarrollo	Lista de correo y BitKeeper

La composición de Linux en cuanto a lenguajes de programación muestra un claro predominio de C, considerado como un lenguaje ideal para la realización de sistemas críticos en cuanto a velocidad. Cuando la velocidad es un requisito tan estricto que ni siquiera C puede alcanzar, se programa directamente en lenguaje ensambla-

ANOTACIONES

dor, un hecho que como podemos ver ocurre con cierta frecuencia. El lenguaje ensamblador tiene la desventaja en comparación con C que no es tan portable, cada arquitectura tiene su juego de instrucciones particular, por lo que mucho código escrito para una arquitectura en ensamblador ha de ser portado a las demás arquitecturas. La presencia del resto de los lenguajes, como se puede observar en el cuadro adjunto, es marginal y se limita a funciones de instalación y utilidades de desarrollo. La versión analizada para este libro ha sido Linux 2.4.21 tal y como fue publicada el 16 de junio del 2003 (sin la aplicación de ningún parche posterior).

Tabla 9-2. Lenguajes de programación utilizados en Linux

Lenguaje de programación	Líneas de código	Porcentaje
C	3062627	94,56%
Ensamblador	164108	5,07%
Shell	3257	0,10%
Yacc	3224	0,10%

9.2. FreeBSD

Como ya se ha comentado en el capítulo dedicado a la historia del software libre, existen otros tipos de sistemas operativos libres además del popular GNU/Linux. Una familia de ellos son los *herederos* de las distribuciones de la Universidad de Berkeley, en California (EE.UU.): los sistemas tipo BSD. El más antiguo y conocido de los sistemas BSD es FreeBSD, cuya historia se remonta a principios de 1993, cuando Bill Jolitz dejó de publicar las actualizaciones no oficiales a 386BSD. Con el apoyo de la empresa Walnut Creek CDROM, que más tarde pasó a llamarse BSDi, un grupo de voluntarios decidió proseguir con el esfuerzo de realizar este sistema operativo libre.

El objetivo principal del proyecto FreeBSD es la creación de un sistema operativo que pueda ser utilizado sin ningún tipo de obligaciones ni ataduras, pero con todas las ventajas de la disponibilidad del código y de un cuidadoso proceso que asegure la calidad del producto. El usuario tiene la libertad de hacer con el software lo que desee, ya

ANOTACIONES

sea modificándolo a su antojo o redistribuirlo –de forma abierta o incluso cerrada y bajo las condiciones que desee– con o sin modificaciones. Como su propio nombre indica, el proyecto FreeBSD se guía, por tanto, por la filosofía de las licencias BSD.

9.2.1. Historia de FreeBSD

La versión de 1.0 apareció a finales de 1993 y estaba basada en 4.3BSD Net/2 y en 386BSD. 4.3BSD Net/2 contaba con código procedente de los años setenta, cuando UNIX era desarrollado por AT&T, lo que a la postre supuso una serie de problemas legales que no se resolvieron hasta que en 1995, FreeBSD 2.0 fue publicado sin contar con código originario de AT&T, esta vez basándose en 4.4BSD-Lite –una versión “light” (se habían suprimido muchos módulos por problemas legales, aparte de que el port para sistemas Intel todavía estaba incompleto) liberada por la Universidad de California de 4.4BSD.

La historia de FreeBSD no sería completa si no se contara nada sobre sus distribuciones “hermanas”, NetBSD y OpenBSD. NetBSD apareció con una versión 0.8 a mediados de 1993. Su principal objetivo era ser muy portable –aunque en sus comienzos sólo fuera un port para i386, por lo que su lema es “por supuesto que corre NetBSD”. OpenBSD surgió de una escisión de NetBSD fundamentada en diferencias filosóficas (y también personales) entre desarrolladores a mediados de 1996. Su principal foco de atención es la seguridad y la criptografía –dicen ser el sistema operativo más seguro que existe–, aunque al basarse en NetBSD también conserva una gran portabilidad.

9.2.2. Desarrollo en FreeBSD

El modelo de desarrollo utilizado por el proyecto FreeBSD está fuertemente basado en dos herramientas: el sistema de versiones CVS y el sistema de informe de error GNATS. Alrededor de estas dos herramientas gira todo el proyecto, como se puede comprobar por el hecho de que se ha creado una jerarquía a partir de las mismas. Así, sobre los *commiters* –aquellos desarrolladores con derecho a escritura al CVS– es sobre quien recae toda la soberanía del proyecto, ya

sea directamente o indirectamente mediante la elección del *core group*, tal como se verá en el siguiente apartado.

Para realizar informes de error en GNATS no hace falta ser *commiter*, por lo que cualquiera que así lo desee podrá notificar una errata. Todas las contribuciones abiertas (*open*) en GNATS son evaluadas por un *commiter*, que podrá asignar (*analized*) la tarea a otro *commiter* o pedir mayor información a la persona que realizó el informe original (*feedback*). Existen situaciones donde la errata ha sido solucionada para algunas ramas recientes, por lo que se especifica con el estado suspendida (*suspended*). En cualquier caso, la meta es que el informe se cierre (*closed*) tras haber corregido el error.

FreeBSD distribuye su software de dos formas. Existen por una parte los *ports*, un sistema que descarga las fuentes, las compila e instala la aplicación en el ordenador local, y por otra están los paquetes, que no son más que las fuentes de los *ports* precompilados y, por tanto, en binario. La ventaja más importante de los *ports* sobre los paquetes es que los primeros permiten al usuario configurar y optimizar el software para su ordenador. Por contra, el sistema de paquetes, al estar ya precompilados, permite instalar el software de una manera más rápida.

9.2.3. Toma de decisiones en FreeBSD

El consejo de directores de FreeBSD, conocido popularmente como el *core team*, se encarga de marcar la dirección del proyecto y de velar porque se cumplan los objetivos del proyecto FreeBSD, así como mediar en caso de que existan conflictos entre *commiters*. Hasta octubre de 2000 era un grupo cerrado al que sólo se entraba a formar parte del mismo por invitación expresa del propio *core team*. A partir de entonces, los miembros del *core team* son elegidos periódica y democráticamente por los *commiters*. La normativa más importante para la elección del *core team* es la siguiente

- Podrán votar los *commiters* que hayan realizado al menos un *commit* en el último año.

- El consejo de directores se renovará cada dos años.

ANOTACIONES

- Los miembros del consejo de directores podrán ser “expulsados” con el voto de dos tercios de los *commiters*.

- Si el número de miembros del consejo de directores es menor de siete, se celebrarán nuevas elecciones.

- Se celebrarán nuevas elecciones si así lo pide un tercio de los *commiters*.

- El cambio de normativa requiere un quórum de dos tercios de los *commiters*.

9.2.4. Empresas alrededor de FreeBSD

Existen multitud de empresas que ofrecen servicios y productos basados en FreeBSD, de las cuales el proyecto FreeBSD lleva buena cuenta en su página del proyecto. En esta presentación de FreeBSD conoceremos un poco más a fondo las más significativas: BSDi y Walnut Creek CDROM.

FreeBSD nació en parte debido a que gente del CSRG (Computer Systems Research Group) de la Universidad de Berkeley fundara en 1991 la compañía BSDi, que se dedicaría al soporte comercial para el nuevo sistema operativo. Además de la versión comercial del sistema operativo FreeBSD, BSDi también desarrolló otros productos como un servidor de Internet y de *gateway*.

Walnut Creek CDROM nació con el objetivo de comercializar FreeBSD como producto final, pudiendo considerarse como una distribución al estilo de las que existen para GNU/Linux, pero con FreeBSD. En noviembre de 1998, Walnut Creek amplió sus horizontes con la creación del portal *FreeBSD Mall* que se dedicaría a comercializar todo tipo de productos sobre FreeBSD (desde la distribución en sí misma, a camisetas, revistas, libros, etc.), a anunciar productos de terceros en su página web y a dar soporte profesional de FreeBSD.

En marzo del 2000, BSDi y Walnut Creek se unieron bajo el nombre BSDi para hacer frente de manera conjunta al fenómeno *Linux,* que

estaba dejando claramente en la sombra a los sistemas BSD en general y a FreeBSD en particular. Un año más tarde, en mayo del 2001, Wind River adquirió la parte dedicada a la generación de software de BSDi, con la clara intención de potenciar el desarrollo de FreeBSD para su uso en sistemas empotrados y dispositivos inteligentes conectados a la Red.

9.2.5. Estado actual de FreeBSD

Según los últimos datos de la encuesta sobre servidores web que realiza periódicamente Netcraft, el número éstos que corre con FreeBSD se acerca a los dos millones de unidades. Un usuario nuevo que quisiera instalarse FreeBSD podría elegir entre la versión 4.8 (la que se podría considerar como la versión "estable") o la más avanzada 5.1 (la de "desarrollo"). Mientras la primera ofrece mayor estabilidad –sobre todo en áreas como el multiprocesamiento simétrico que han sido totalmente reelaboradas en las nuevas versiones–, la segunda permite disfrutar de las últimas novedades. También es importante tener en cuenta que las versiones de desarrollo suelen incluir código de pruebas, lo que hace que la velocidad del sistema se vea afectada sensiblemente.

Una de las utilidades estrella de FreeBSD son las llamadas *cárceles* (*jail* en inglés). Las cárceles permiten minimizar el daño causado por un ataque a servicios de red básicos como podrían ser sendmail para el correo electrónico o BIND (*Berkeley Internet Name Domain*) para gestionar los nombres. Los servicios son introducidos en una cárcel para que se ejecuten en un entorno de aislamiento. La gestión de las cárceles se puede realizar mediante una serie de utilidades incluidas en FreeBSD.

9.2.6. Radiografía de FreeBSD

Como se ha venido comentando a lo largo de este último apartado, la labor de BSD no se restringe únicamente al desarrollo de un núcleo del sistema operativo, sino que también incluye la integración de multitud de utilidades que se distribuyen conjuntamente al estilo de las distribuciones de GNU/Linux. El hecho de que el proceso de desarrollo de FreeBSD esté íntimamente ligado al sistema de control de

ANOTACIONES

versiones CVS hace que el estudio del mismo nos pueda dar una buena aproximación de todo lo que contiene. Las cifras que se muestran a continuación son las correspondientes al análisis de FreeBSD efectuado el 21 de agosto del 2003.

Uno de los aspectos más interesantes de FreeBSD es que sus números se asemejan mucho a los que ya hemos podido observar en KDE y en GNOME: el tamaño del software supera ampliamente los cinco millones de líneas de código, el número de ficheros ronda los 250.000 y el número total de *commits* se sitúa en torno a los dos millones. Sin embargo, es interesante observar que la principal diferencia entre GNOME y KDE con FreeBSD es la *edad* del proyecto. FreeBSD acaba de cumplir hace escasas fechas la década de existencia y casi dobla en tiempo a los entornos de escritorio con los que la estamos comparando. Que el tamaño sea similar, aun cuando el tiempo de desarrollo ha sido superior, se debe en gran parte a que el número de desarrolladores que ha atraído FreeBSD es más pequeño. Con derecho de escritura en el CVS (*commiter*) hay listados unos 400, mientras que los colaboradores que se mencionan en el manual de FreeBSD son cerca de un millar. Por esta razón, la actividad que registra el CVS de FreeBSD es menor en media (500 *commits* diarios) que la que registraban tanto GNOME (900) como KDE (1700 contando los *commits automáticos*).

Hemos considerado como sistema básico de FreeBSD todo aquello que cuelga del directorio src/src del módulo root del CVS. La actividad que ha venido registrando el sistema básico a lo largo de los últimos diez años es de más de medio millón de *commits*. Su tamaño supera los cinco millones de líneas de código, aunque hay que comentar que no sólo está incluido el kernel en él, sino multitud de utilidades adicionales, incluso juegos. Si tenemos en cuenta sólo el kernel (que se encuentra bajo el subdirectorio sys), su tamaño es de 1,5 millones de líneas de código fuente, predominantemente en C.

Resulta interesante ver cómo la estimación de tiempo dada por COCOMO concuerda a la perfección con el tiempo real que ha llevado el proyecto FreeBSD, aunque la estimación de número de desarrolladores medio es, en mucho, mayor que la real. Nótese que en el último año, sólo unos 75 *commiters* han estado activos, mientras que

ANOTACIONES

COCOMO supone que durante los 10 años de desarrollo el número de desarrolladores debería ser de 235.

Por último, cabe destacar, como se ha hecho anteriormente, que la actividad principal de FreeBSD se sitúa en torno al CVS y al sistema de control de erratas y actividades GNATS.

Tabla 9-3. Estado actual de FreeBSD

Página web	http://www.FreeBSD.org
Inicio del proyecto	1993
Licencia	tipo BSD
Versión actual de FreeBSD	4.8 (estable), 5.1 (en desarrollo)
Líneas de código fuente	7.750.000
Líneas de código fuente (sólo núcleo)	1.500.000
Número de ficheros	250.000
Estimación de coste	$ 325.000.000
Estimación de tiempo de ejecución	10,5 años (126 meses)
Estimación de número medio de desarrolladores	235
Número aproximado de desarrolladores	400 *commiters* (1000 colaboradores)
Número de *commiters* activos en el último año	75 (menos 20% del total)
Número de *commiters* activos en los dos últimos años	165 (alrededor del 40% del total)
Número de *commits* en el CVS	2.000.000
Número medio de *commits* (totales) al día	Unos 500
Herramientas de ayuda al desarrollo	CVS, GNATS, listas de correo y sitio de noticias

C es el lenguaje predominante en FreeBSD, siendo la distancia con C++ superior a la de los demás casos que hemos estudiado en este capítulo. Es interesante observar cómo el número de líneas de código ensamblador contenidos en FreeBSD concuerdan en el orden de magnitud con las que tiene Linux, aunque las correspondientes al núcleo sólo sumen unas 25.000 líneas en total. En resumen, se podría decir que en FreeBSD lo que manda son los lenguajes más *clásicos* dentro del software libre, C, shell y Perl, y que la penetración de los lenguajes que hemos venido observando en otras aplicaciones y proyectos –C++, Java, Python, etc.– no ha tenido lugar.

ANOTACIONES

Tabla 9-4. Lenguajes de programación utilizados en FreeBSD

Lenguaje de programación	Líneas de código	Porcentaje
C	7.080.000	92,0%
Shell	205.000	2,7%
C++	131.500	1,7%
Ensamblador	116.000	1,5%
Perl	90.900	1,20%
Yacc	5.800	0,75%

9.2.7. Estudios académicos sobre FreeBSD

Lectura complementaria

[Ver bibliografía, apartado 129]

Aun siendo ciertamente un proyecto muy interesante –podemos obtener toda su historia (¡desde hace 10 años!) mediante el análisis del sistema de versiones–, la atención que ha despertado FreeBSD entre la comunidad científica ha sido más bien pequeña. De esta falta de interés se salva un equipo de investigación, que ha estudiado el proyecto FreeBSD desde varios puntos de vista, poniendo especial atención a cómo se resuelven los problemas de la integración de software de manera incremental y descentralizada.

9.3. KDE

Aunque con probabilidad no fue la primera solución en cuanto a entornos de escritorios *amigables* para el usuario, la difusión a mediados de 1995 del sistema operativo Windows95(TM) supuso un cambio radical en la interacción de los usuarios de a pie con los ordenadores. De los sistemas unidimensionales de línea de instrucciones (los terminales), se pasó a la metáfora de entorno del escritorio bidimensional, donde el ratón ganó terreno al teclado. Windows95(TM), más que una innovación tecnológica, debe ser acreditado como el sistema que consiguió adentrarse en todos los entornos personales y de oficina, marcando las pautas a seguir (reglas técnicas y sociales que, a principios del siglo XXI, a veces todavía seguimos padeciendo).

Con anterioridad a los sistemas de escritorio, cada aplicación gestionaba la apariencia y la forma de interactuar con el usuario de ma-

ANOTACIONES

nera autónoma. En los escritorios, por el contrario, las aplicaciones han de contar con propiedades comunes y un aspecto compartido entre aplicaciones, de forma que esto suponga un alivio para el usuario, que puede *reutilizar la interacción* aprendida de una aplicación en otra aplicación. También resultó ser un alivio para los desarrolladores de aplicaciones, ya que no se tenían que enfrentar con la problemática de realizar los elementos interactivos desde cero (una tarea siempre complicada), sino que podían partir de un marco y unas reglas predefinidas.

9.3.1. Historia de KDE

Rápidamente, los seguidores de UNIX se hicieron eco del notable éxito de Windows 95 y, a la vista de que los entornos UNIX carecían de sistemas tan intuitivos a la vez que libres, decidieron ponerse manos a la obra. Fruto de esta preocupación nació en 1996 el proyecto KDE –K Desktop Environment– de las manos de Matthias Ettrich (creador de LyX, un programa de edición en modo gráfico de TeX) y otros *hackers*. El proyecto KDE se planteó los siguientes objetivos:

- Dotar a los sistemas UNIX de un entorno amigable que sea al mismo tiempo abierto, estable, de confianza y poderoso.
- Desarrollar un conjunto de bibliotecas para escribir aplicaciones estándar sobre el sistema gráfico para UNIX X11.
- Crear una serie de aplicaciones que permitan al usuario acometer sus objetivos de manera eficaz.

Nota

Originariamente, el nombre KDE significaba *Kool Desktop Environment*, pero con el tiempo se decidió que pasara a llamarse simplemente *K Desktop Environment*, siendo la explicación oficial que la K es la letra que precede en el alfabeto latino a la L de Linux.

Cuando los integrantes del recién creado proyecto KDE decidieron utilizar una biblioteca orientada a objetos llamada Qt, propiedad de la firma noruega Trolltech(TM), que no estaba amparada bajo una

Lectura complementaria

[Ver bibliografía, apartado 95]

ANOTACIONES

licencia de software libre, se originó una gran polémica. Se daba, por tanto, la circunstancia de que, a pesar de que las aplicaciones de KDE estaban licenciadas bajo la GPL, enlazaban con esta biblioteca de manera que se hacía imposible su redistribución. Consecuentemente, se estaba violando una de las cuatro libertades del software libre enunciadas por Richard Stallman en el *Manifiesto del software libre*. Desde la versión 2.0, Trolltech distribuye Qt bajo una licencia dual que especifica que si la aplicación que hace uso de la biblioteca es GPL, entonces la licencia válida para Qt es la GPL. Gracias a ello, uno de los debates más calientes y airados dentro del mundo del software libre tuvo, por suerte, un final feliz.

9.3.2. Desarrollo de KDE

KDE es de los pocos proyectos de software libre que cumplen con un calendario de lanzamiento de nuevas versiones de manera generalizada (recordemos, por ejemplo, que hay una nueva versión de Linux, *cuando esté lista*, mientras que, como veremos más adelante, en el caso de GNOME se han dado retrasos significativos a la hora de publicar nuevas versiones). La numeración de las versiones sigue una política perfectamente definida: las versiones de KDE constan de tres números de versión: uno mayor y dos menores. Por ejemplo, en KDE 3.1.2, el número mayor sería el 3 y los menores el 1 y el 2. Versiones con el mismo número mayor cuentan con compatibilidad binaria, por lo que no hace falta recompilar las aplicaciones. Hasta ahora, los cambios en el número mayor se han venido dando paralelamente con cambios en la biblioteca Qt, por lo que se puede ver que los desarrolladores quisieron aprovechar las nuevas funcionalidades de la biblioteca Qt en la versión inminente de KDE.

En cuanto a los números menores, las versiones con un único número menor son versiones en las que se han incluido tanto nuevas funcionalidades, como corrección de las erratas encontradas. Las versiones con segundo número menor no incluyen nuevas funcionalidades sobre las versiones con primer número menor y sólo contienen corrección de errores. Para aclararlo con un ejemplo: KDE 3.1 es una versión de la tercera generación de KDE (número mayor 3) a la que se le han añadido nuevas funcionalidades, mientras KDE 3.1.1 es la versión anterior –con las mismas funcionalidades–, pero con las erratas que se han encontrado corregidas.

ANOTACIONES

KDE se constituyó poco después de empezar en una asociación registrada en Alemania (KDE e.V.) y como tal tiene unos estatutos que la obligan a contar con un consejo directivo. La influencia del consejo directivo sobre el desarrollo es nula, ya que su tarea es fundamentalmente la administración de la asociación, en especial de las donaciones que percibe el proyecto. Para la promoción y difusión de KDE –incluyendo a empresas interesadas– se creó la Liga KDE, que se introducirá a continuación.

9.3.3. La Liga KDE

La Liga KDE (KDE League en su denominación original en inglés) es un grupo de empresas y de particulares de KDE con el objetivo de facilitar la promoción, distribución y desarrollo de KDE. Las empresas y particulares que participan en la Liga KDE no tienen que estar directamente involucrados en el desarrollo de KDE, aunque se anima a todos los miembros a hacerlo, sino que simplemente representan un marco industrial y social amigo de KDE. Los objetivos de la Liga KDE son los siguientes:

- Promover, proveer y facilitar la educación formal e informal de las funcionalidades, capacidades y otras cualidades de KDE.
- Animar a corporaciones, gobiernos, empresas e individuos a usar KDE.
- Animar a corporaciones, gobiernos, empresas e individuos a participar en el desarrollo de KDE.
- Proveer conocimientos, información, dirección y posicionamiento alrededor de KDE en cuanto a su uso y su desarrollo.
- Promocionar la comunicación y la cooperación entre los desarrolladores de KDE.
- Promocionar la comunicación y la cooperación entre los desarrolladores de KDE y el público mediante publicaciones, artículos, sitios web, encuentros, participación en congresos y exposiciones, notas de prensa, entrevistas, material promocional y comités.

ANOTACIONES

Las empresas que participan en la KDE League son principalmente distribuciones (SuSE, Mandrake, TurboLinux, Lindows y Hancom –una distribución de software libre coreana), empresas de desarrollo (Trolltech y Klarälvdalens Datakonsult AB), además del gigante IBM y una empresa creada con la finalidad de promocionar KDE (KDE.com). De entre todas ellas, cabe destacar por su implicación fundamental Trolltech, SuSE y Mandrake software, cuyo modelo de negocio está íntimamente ligado al proyecto KDE:

- Trolltech es una compañía noruega afincada en Oslo que desarrolla Qt, la biblioteca que hace las veces de interfaz gráfica de usuario y API para el desarrollo de aplicaciones, aunque también puede funcionar como elemento empotrado en PDA (como por ejemplo en los Sharp Zaurus). La importancia del proyecto KDE en Trolltech se puede constatar en dos elementos muy importantes de su estrategia comercial. Por un lado reconoce a KDE como su principal forma de promoción, alentando al desarrollo del escritorio y aceptando e implementando las mejoras o modificaciones propuestas. Por otro lado, algunos de los desarrolladores más importantes de KDE trabajan profesionalmente para Trolltech –el caso más conocido es el del propio Mathias Ettrich, fundador del proyecto– lo que, sin duda, beneficia tanto al proyecto KDE, como a la propia compañía. La implicación de Trolltech en el proyecto KDE no se limita exclusivamente a la biblioteca Qt, como se puede ver por el hecho de que uno de los desarrolladores principales de KOffice –la suite de oficina de KDE– tenga en la actualidad un contrato a tiempo parcial con ellos.

- SuSE siempre ha demostrado una especial predilección por el sistema de escritorio KDE, en parte debido a que una gran mayoría de los desarrolladores son de origen alemán o centro-europeo, al igual que la propia compañía. Conocedora del hecho de que cuanto mejor y más fácil sea el entorno de escritorio que ofrezca su distribución, mayor será su implantación y, por tanto, las ventas y la petición de soporte, SuSE ha practicado siempre una política muy activa en cuanto a dedicar presupuesto para profesionalizar posiciones clave dentro del proyecto KDE. De esta manera, en la actualidad el administrador del sistema de control de versiones y otro par de desarrolladores principales cuentan con una nómina de SuSE. Asimismo, dentro de la plantilla de SuSE existe una

ANOTACIONES

docena de desarrolladores que pueden dedicar parte de su tiempo laboral al desarrollo de KDE.

- La distribución Mandrake es otro de los grandes benefactores de KDE, contando entre su plantilla con varios de los desarrolladores principales. Su situación económica en el 2003 – con suspensión de pagos incluida– ha hecho que en los últimos tiempos haya perdido en influencia.

9.3.4. Estado actual de KDE

Tras la publicación de KDE 3 en mayo del 2002, la opinión generalizada es que los escritorios libres se encuentran a la altura de sus competidores propietarios. Entre sus grandes logros se encuentra la incorporación de un sistema de componentes (KParts) que permite empotrar unas aplicaciones en otras (un trozo de hoja de cálculo de KSpread en el procesador de textos KWord) y el desarrollo de DCOP, un sistema de comunicación entre procesos simple y con autentificación. DCOP fue la apuesta del proyecto en detrimento de las tecnologías CORBA, un tema de amplio debate dentro de los escritorios libres, en especial entre GNOME –que se decidió por utilizar tecnologías CORBA– y KDE. La historia parece haber puesto a cada tecnología en su sitio, como se puede ver con la propuesta de DBUS (una especie de DCOP mejorado) por parte del FreeDesktop.org, un proyecto interesado en fomentar la interoperabilidad y el uso de tecnologías conjuntas en los escritorios libres, que casualmente está liderado por uno de los *hackers* de GNOME más reconocidos.

El siguiente cuadro resumen contiene las características más importantes del proyecto KDE. Las licencias que acepta el proyecto depende de si se trata de una aplicación o una biblioteca. Las licencias de las bibliotecas permiten una mayor “flexibilidad” a terceros; en otras palabras, posibilita que terceros puedan crear aplicaciones propietarias que enlacen con las bibliotecas.

La versión actual de KDE es la 3.1.3, aunque se está trabajando activamente en la rama de la siguiente versión con la inclusión de nuevas funcionalidades, KDE 3.2. Para el 2004 se espera la cuarta generación de KDE, KDE 4, que se basará en Qt4. Los cambios de generación suponen un gran esfuerzo de adaptación, una tarea te-

ANOTACIONES

diosa y costosa en tiempo. Sin embargo, esto no ha de suponer que las aplicaciones "antiguas" dejen de funcionar. Generalmente, para que sigan funcionando se incluyen también las antiguas versiones de las bibliotecas sobre las que se basa, aunque ello suponga tener que cargar varias versiones de las bibliotecas en memoria simultáneamente con el consiguiente desperdicio de recursos del sistema. Este hecho es visto por los desarrolladores de KDE como algo inherente a la propia evolución del proyecto y, por tanto, como un mal menor.

9.3.5. Radiografía de KDE

En cuanto al tamaño de KDE, las cifras que se van a mostrar a continuación corresponden al estado del CVS en agosto del 2003, por lo que han de tomarse con las precauciones tradicionales que ya se han comentado y una más: alguno de los módulos que se han considerado en este estudio todavía se encuentran en fase de desarrollo y no cumplen con el requisito de ser un producto acabado. Esto no debe suponer un obstáculo para nuestros propósitos, ya que estamos más bien interesados en el orden de magnitud de los resultados más que en la cifra exacta.

El código fuente incluido en el CVS de KDE suma en total más de seis millones de línea de código en diferentes lenguajes de programación, tal como se mostrará más adelante. El tiempo que se necesitaría para crear KDE rondaría los nueve años y medio, una cifra superior a los siete años con los que cuenta el proyecto, y el número medio de desarrolladores a tiempo completo estimado rondaría los 200. Si tenemos en cuenta que KDE cuenta con unas 800 personas con acceso de escritura en el CVS en el 2003 (de los cuales la mitad ha estado inactivo en los últimos dos años) y que el número de desarrolladores de KDE contratados a tiempo completo no ha superado en ningún momento la veintena, podemos ver que la productividad de KDE es, en mucho, superior a la estimación que ofrece COCOMO.

Nota

Una empresa que quisiera desarrollar un producto de ese tamaño desde cero necesitaría invertir más de 250 millones de dólares, una cifra que, a modo de comparación, es lo que invirtió una firma de automóviles en

ANOTACIONES

> la creación de una nueva planta de producción en el este de Europa o lo que una conocida petrolera planea gastar para abrir 200 gasolineras en España.

Es interesante ver que una gran parte del esfuerzo –casi la mitad que el del desarrollo– del proyecto KDE lo podemos situar en la traducción de la interfaz de usuario y de la documentación. Aunque muy pocas (unas miles) de las líneas de programación se dedican a esta labor, el número de ficheros dedicados a este cometido asciende a los 75.000 para traducciones (cifra que se eleva hasta los 100.000 si incluimos la documentación en sus diferentes formatos), lo que viene a suponer casi la cuarta (tercera) parte de los 310.000 ficheros que hay en el CVS. La actividad conjunta del CVS es de 1.200 *commits* diarios, por lo que el tiempo medio entre *commits* es de cerca de un minuto.

En cuanto a las herramientas, lugares de información y eventos de ayuda al desarrollo, vemos que el abanico de posibilidades que ofrece KDE es mucho más amplio que el usado en Linux. Además del sistema de control de versiones y de las listas de correo, KDE cuenta con una serie de sitios web donde se puede encontrar información y documentación técnica y no técnica del proyecto. Entre estos sitios también existe un sitio de noticias, donde se presentan nuevas soluciones y se debaten propuestas. El sitio de noticias, sin embargo, no puede considerarse como sustituto de las listas de correo –que, al igual que en Linux, es donde se encuentran las verdaderos debates y la toma de decisiones y estrategias de futuro–, sino como un punto de encuentro con los usuarios. Por último, KDE organiza desde hace tres años reuniones periódicas, donde los desarrolladores y colaboradores se juntan durante aproximadamente una semana para presentar las últimas novedades, desarrollar, debatir, conocerse y pasárselo bien (no necesariamente en este orden).

Tabla 9-5. Estado actual de KDE

Página web	http://www.kde.org
Inicio del proyecto	1996
Licencia (para aplicaciones)	GPL, QPL, MIT, Artistic
Licencia (para bibliotecas)	LGPL, BSD, X11

ANOTACIONES

Versión actual de KDE	3.1.3
Líneas de código fuente	6.100.000
Número de ficheros (código, documentación, etc.)	310.000 ficheros
Estimación de coste	$ 255.000.000
Estimación de tiempo de ejecución	9,41 años (112.98 meses)
Estimación de número medio de desarrolladores	200,64
Número aproximado de desarrolladores	Unos 900 *commiters*
Número de *commiters* activos en el último año	Alrededor de 450 (aproximadamente el 50% del total)
Número de *commiters* activos en los dos últimos años	Unos 600 (65% del total)
Número aproximado de traductores (activos)	Unos 300 traductores para más de 50 lenguas (incluido el esperanto)
Número de *commits* (de desarrolladores) en el CVS	Aproximadamente 2.000.000 (cifra estimada sin *commits* automáticos)
Número de *commits* (de traductores) en el CVS	Aproximadamente 1.000.000 (cifra estimada sin *commits* automáticos)
Número medio de *commits* (totales) al día	1.700
Herramientas, documentación y eventos de ayuda al desarrollo	CVS, listas de correo, sitio web, sitio de noticias, reuniones anuales, etc.

En cuanto a los lenguajes de programación utilizados en KDE, predomina el uso de C++. Esto se debe principalmente a que éste es el lenguaje nativo de Qt, aunque se realiza un gran esfuerzo en realizar enlaces para permitir el desarrollo en otros lenguajes de programación. Ciertamente, el número de líneas de código en los lenguajes minoritarios corresponde casi íntegramente al propio proyecto de creación del enlace, aunque esto no quiere decir que no se utilice en absoluto, ya que existen multitud de proyectos externos a KDE que los utilizan.

Tabla 9-6. Lenguajes de programación utilizados en KDE

Lenguaje de programación	Líneas de código	Porcentaje
C++	5011288	82,05%
C	575237	9,42%
Objective C	144415	2,36%
Shell	103132	1,69%
Java	87974	1,44%
Perl	85869	1,41%

ANOTACIONES

9.4. GNOME

El proyecto GNOME tiene como principal objetivo crear un sistema de escritorio para el usuario final que sea completo, libre y fácil de usar. Asimismo, se pretende que GNOME sea una plataforma muy potente de cara al desarrollador. GNOME es el acrónimo en inglés de *GNU Network Object Model Environment*. Desde los inicios de GNOME se han propuesto varias formas de traducirlo al castellano, pero no se ha encontrado ninguna que haya satisfecho a todos. Sin embargo, de su nombre podemos ver que GNOME es parte del proyecto GNU. En la actualidad, todo el código contenido en GNOME debe estar bajo licencia GNU GPL o GNU LGPL. También vemos que las redes y el modelado orientado a objetos tienen capital importancia.

9.4.1. Historia de GNOME

Mientras se seguía discutiendo acerca de la libertad de KDE, la historia quiso que en el verano de 1997, Miguel de Icaza y Nat Friedman coincidieran en Redmond en unas jornadas organizadas por Microsoft(TM). Es probable que este encuentro propiciara en ambos un giro radical que supuso tanto la creación de GNOME por parte de Miguel de Icaza a su vuelta a México (junto con Federico Mena Quintero), como su admiración por las tecnologías de objetos distribuidos. Icaza y Mena decidieron crear un entorno alternativo a KDE, ya que consideraron que una reimplementación de una biblioteca propietaria hubiera sido una tarea destinada a fracasar. Había nacido GNOME.

Desde aquellos tiempos lejanos de 1997 hasta la actualidad, GNOME ha ido creciendo paulatinamente con sus reiteradas publicaciones. En noviembre de 1998 se lanzó la versión 0.99, pero la primera realmente popular distribuida prácticamente por cualquier distribución de GNU/Linux sería GNOME 1.0, en marzo de 1999. Cabe destacar que la experiencia de esta primera versión *estable* de GNOME no fue muy satisfactoria, ya que muchos la consideraron llena de erratas críticas. Por eso, GNOME October (GNOME 1.0.55) es tratada como la primera versión del entorno de escritorio GNOME realmente estable. Como se puede observar, con GNOME October se

ANOTACIONES

intentó evitar versiones de publicación numeradas para no entrar en una *carrera* de versiones con KDE. La realización de la primera GUADEC, la conferencia de desarrolladores y usuarios europeos de GNOME, celebrada en París en el año 2000, no coincidió en el tiempo por poco con la publicación de una nueva versión de GNOME, llamada GNOME April. Fue la última que llevó un mes como nombre de publicación, ya que se mostró que ese sistema causaba más confusión que otra cosa (por ejemplo, GNOME April es posterior a GNOME October, aunque el sentido común nos haría pensar lo contrario). En octubre de ese año, tras ser debatida durante meses en diferentes listas de correo, se fundó la Fundación GNOME que será presentada más adelante.

GNOME 1.2 fue un paso adelante en cuanto a la arquitectura utilizada por GNOME, que se siguió usando en GNOME 1.4. Esta época estuvo caracterizada por la segunda edición de la GUADEC, esta vez en Copenhague. Lo que empezó siendo una reunión minoritaria de algunos *hackers*, se había convertido en un gran evento que atrajo miradas de toda la industria del software.

Mientras tanto, el litigio sobre la libertad de KDE se resolvió con el cambio de postura de Trolltech(TM), que terminó licenciando Qt bajo una licencia dual, que era de software libre para las aplicaciones que son software libre. Hoy en día no cabe ninguna duda de que tanto GNOME como KDE son entornos de escritorio libres, por lo que podemos considerar que el desarrollo de GNOME ha propiciado el hecho de no tener un solo entorno de escritorio libre, sino dos.

9.4.2. La Fundación GNOME

El problema más difícil de abordar cuando se oye hablar de GNOME por primera vez, es la organización de los más de 1.000 contribuyentes al proyecto. Resulta paradójico que un proyecto cuya estructura es más bien anárquica, llegue a fructificar y saque adelante unos objetivos complejos, que están al alcance de pocas multinacionales del sector de la informática.

Aunque GNOME nació con una clara intención de realizar un entorno amigable y potente al que se iban añadiendo nuevos programas, pronto se vio la necesidad de crear un órgano que tuviera ciertas

ANOTACIONES

competencias que permitieran potenciar el uso, desarrollo y difusión de GNOME: de esta forma, en octubre de 2000 se dio paso a la creación de la Fundación GNOME, cuya sede se encuentra en Boston, EE.UU.

La Fundación GNOME es una organización sin ánimo de lucro, no un consorcio industrial, que tiene las siguientes funciones:

- Coordina las publicaciones.
- Decide qué proyectos son parte de GNOME.
- Es la voz oficial (para la prensa y para organizaciones tanto comerciales como no comerciales) del proyecto GNOME.
- Patrocina conferencias relacionadas con GNOME (como la GUADEC).
- Representa a GNOME en otras conferencias.
- Crea estándares técnicos.
- Promueve el uso y el desarrollo de GNOME.

Además, la Fundación GNOME permite la recepción de fondos económicos con los cuales patrocinar e impulsar las funciones antes mencionadas, hecho que antes de su creación era imposible realizar de manera transparente.

En la actualidad, la Fundación GNOME cuenta con un empleado a tiempo completo que se encarga de solventar todos los trabajos burocráticos y organizativos que se dan en una organización sin fines de lucro que realiza reuniones y conferencias de manera periódica.

En términos generales, la Fundación GNOME se estructura en dos grandes consejos: el consejo directivo y el consejo consultor.

El consejo directivo (Board of Directors) está integrado a lo sumo por catorce miembros elegidos democráticamente por los miembros de

la Fundación GNOME. La membresía sigue un modelo meritocrático, lo que viene a decir que para ser miembro de la Fundación GNOME se debe de haber colaborado de alguna u otra manera con el proyecto GNOME. La aportación no tiene por qué ser necesariamente código fuente, también existen tareas de traducción, organización, difusión, etc., por las que uno puede solicitar ser miembro de la Fundación GNOME y tener derecho a voto. Por tanto, son los miembros de la Fundación los que se pueden presentar al Consejo Directivo y los que, democráticamente, eligen a sus representantes en el mismo de entre los que se hayan presentado. En la actualidad, la votación se lleva a cabo por correo electrónico. La duración del cargo como consejero director es de un año, periodo tras el que se vuelven a convocar elecciones.

Existen unas normas básicas para garantizar la transparencia del consejo directivo. La más llamativa es la limitación de miembros afiliados a una misma empresa, que no puede exceder de cuatro empleados. Es importante hacer hincapié en que los miembros del Consejo Directivo lo son siempre a nivel personal y nunca en representación de una compañía. Aun así, y después de una larga discusión, se aceptó incluir esta cláusula para evitar suspicacias.

El otro consejo dentro de la Fundación GNOME es el Consejo Consultor. El Consejo Consultor es un órgano sin capacidad de decisión que sirve como vehículo de comunicación con el Consejo Directivo. Está compuesto por compañías comerciales de la industria del software así como por organizaciones no comerciales. En la actualidad sus miembros son Red Hat, Ximian, Hewlett-Packard, Mandrake, SUN Microsystems, Red Flag Linux, Wipro, Debian y la Free Software Foundation. Para formar parte del Consejo de Consultores se exige una cuota a todas las empresas con más de 10 empleados.

9.4.3. La industria alrededor de GNOME

GNOME ha conseguido adentrarse de manera sustancial en la industria, de manera que varias empresas han participado muy activamente en su desarrollo. De todas ellas, los casos más significativos son los de Ximian Inc., Eazel, los RHAD Labs de Red Hat y, más recientemente, SUN Microsystems. A continuación se describe para

ANOTACIONES

cada caso tanto las motivaciones de las compañías, como sus aportaciones más importantes al entorno de escritorio GNOME:

- Ximian Inc. (en sus comienzos Helix Inc.) es el nombre de la empresa que fundaron en 1999 Miguel de Icaza, cofundador de GNOME, y Nat Friedman, uno de los *hackers* de GNOME. Su principal acometido era reunir bajo un mismo paraguas a los desarrolladores más importantes de GNOME para potenciar su desarrollo, por lo que no es de extrañar que cuente o haya contado entre sus empleados con una veintena larga de los desarrolladores más activos de GNOME. La aplicación en la que Ximian puso mayor empeño desde sus comienzos ha sido Evolution, un completo sistema de gestión de información personal al estilo de Microsoft Outlook que incluya cliente de correo-e, agenda y directorio de contactos. Los productos que Ximian comercializaba son el Ximian Desktop (una versión de GNOME con fines más corporativos), RedCarpet (un sistema de distribución de software, principalmente, aunque no exclusivamente, de GNOME) y finalmente MONO (una reimplementación de la plataforma de desarrollo .NET), aunque este último proyecto, por ahora, no tenga nada que ver con GNOME. Ximian también ha desarrollado una aplicación que sirve para que Evolution interactúe con un servidor Exchange 2000. Esta aplicación, aunque bastante pequeña, fue muy polémica porque se publicó con una licencia no libre. En agosto del 2003 Novell, como parte de su estrategia para entrar en el escritorio de GNU/Linux, compró Ximian.

- Eazel fue fundada en 1999 por un grupo de personas proveniente de Apple con la finalidad de hacer el escritorio en GNU/Linux tan fácil como lo es en Macintosh. La aplicación en la que centraron su esfuerzo recibió el nombre de Nautilus y debía ser el gestor de ficheros que jubilara el mítico Midnight Commander desarrollado por Miguel de Icaza. Su falta de modelo de negocio y la crisis de las puntocom –los inversores de riesgo retiraron el capital necesario para que Eazel pudiera seguir funcionando– provocó que el 15 de mayo del 2001, Eazel se declarara en quiebra y cerrara sus puertas. Todavía tuvo tiempo para publicar la versión 1.0 de Nautilus, aunque su numeración fuera más bien artificial, ya que la estabilidad que se presupone a una versión 1.0 no aparecía por ningún lado. Dos años después de la quiebra de Eazel, se puede ver cómo Nautilus ha evolucionado y se ha convertido en

ANOTACIONES

un completo y manejable gestor de ficheros integrado en GNOME, por lo que la historia de Eazel y Nautilus se puede considerar como un caso paradigmático de un programa que sobrevive a la desaparición de la empresa que lo creaba –algo casi sólo posible en el mundo del software libre.

- Red Hat creó los Red Hat Advanced Development Labs (laboratorios de desarrollo avanzado de Red Hat), RHAD, con la intención de que el escritorio GNOME ganara en usabilidad y potencia. Para ello, contrató a media docena de los *hackers* más importantes de GNOME y les dio libertad para desarrollar en lo que ellos decidieran que era conveniente. De los RHAD Labs salió ORBit, la implementación de CORBA utilizada por el proyecto GNOME, conocida como la más rápida del oeste. También es destacable la labor que se llevó a cabo en la nueva versión de GTK+ y en el sistema de configuración de GNOME, GConf.

- SUN Microsystems se involucró tardíamente en el desarrollo de GNOME, ya que para septiembre del 2000 GNOME era un producto relativamente maduro. La intención de SUN era utilizar GNOME como el sistema de escritorio del sistema operativo Solaris. Para ello, creó un equipo de colaboración con GNOME, cuyos méritos más importantes giran en torno a la usabilidad y la accesibilidad de GNOME. En junio del 2003, SUN anunció que distribuiría GNOME 2.2 con la versión 9 de Solaris.

9.4.4. Estado actual de GNOME

GNOME se encuentra en la actualidad en su versión 2.2, camino a la siguiente, la 2.4. La mayoría de las tecnologías clave en las que se basa se encuentran maduras como se puede desprender de su número de versión. Así, el *broker* CORBA utilizado es ORBit2, mientras que el entorno gráfico y API, GTK+, acogió los cambios fruto de la experiencia acumulada en las versiones anteriores de GNOME. Como gran novedad, aparece la inclusión de una biblioteca de accesibilidad propuesta por SUN, que permite que las personas que tengan problemas de accesibilidad puedan utilizar el entorno GNOME. Mención especial también requiere bonobo, el sistema de componentes de GNOME. Bonobo marcó una época dentro de GNOME a la par que se iba desarrollando el gestor de información personal

ANOTACIONES

Evolution. Sin embargo, el tiempo ha demostrado que las expectativas levantadas por bonobo fueron demasiado altas y la reutilización de esfuerzo mediante el uso de componentes no ha sido la que en un principio se esperaba.

Nota

La biblioteca ATK es una biblioteca de clases abstractas que permite hacer las aplicaciones accesibles. Esto quiere decir que las personas con alguna discapacidad (ciegos, daltónicos, gente con problemas de vista, que no pueden manejar el ratón o el teclado, etc.) puedan hacer uso de GNOME. El interés de SUN por la accesibilidad parte de que para poder ofrecer sus productos al Gobierno de Estados Unidos ha de cumplir una serie de estándares en cuanto a accesibilidad. Tan en serio se lo ha tomado, que el equipo de desarrollo de GNOME que trabaja en SUN cuenta incluso con un programador invidente. La arquitectura de accesibilidad de GNOME recibió en septiembre del 2002 el premio Hellen Keller Achievement Award.

9.4.5. Radiografía de GNOME

Los datos y las cifras que se muestran en la tabla 9-7 nos permitirán cerrar la presentación de GNOME. Las cifras que se muestran corresponden al estado del CVS de GNOME el 14 de agosto del 2003. Ese día había más de nueve millones de líneas de código hospedadas en el repositorio CVS que tiene el proyecto GNOME. Aun cuando una comparación con KDE sería lo más natural, hemos de advertir al lector que las diferencias en cuanto a la organización de los proyectos la hacen desaconsejable si se quiere hacer en igualdad de condiciones. Esto se debe, por ejemplo, a que el CVS de GNOME incluye GIMP (un programa de creación y manipulación de gráficos), responsable él solo de más de 660 mil líneas de código, o de la biblioteca GTK+, sobre la que se centra el desarrollo en GNOME, que cuenta a su vez con 330 mil líneas. Si a esto se le añade que el repositorio CVS de GNOME es más proclive a abrir nuevos módulos para programas (en total cuenta con 700) que el de KDE (que cuenta con menos de 100), podremos entender por qué GNOME cuenta con un número de líneas superior al de KDE a pesar de ser un año y

ANOTACIONES

medio más joven. El repositorio de GNOME acoge a más de 225 mil ficheros que han sido añadidos y modificados casi dos millones de veces (véase el número de *commits* unas filas más abajo en la tabla).

Nota

Una empresa cuyo deseo fuera crear un software del tamaño de GNOME debería contratar de media a unos 250 desarrolladores durante más de once años para conseguir un producto de tamaño similar, según el modelo COCOMO utilizado a lo largo de todo este capítulo. El coste asociado ascendería a unos 400 millones de dólares, una cifra parecida a la que una empresa de telefonía móvil ya asentada invertirá en el 2003 para reforzar su capacidad de red, o similar a la que desembolsará una compañía de automóviles para abrir una planta de producción en Barcelona.

GNOME cuenta con unos recursos humanos de casi mil desarrolladores con acceso de lectura al sistema de control de versiones CVS, contando con casi una veintena de desarrolladores que se dedican a GNOME de manera profesional (a tiempo completo o tiempo parcial). De éstos, sólo un 25% se ha mostrado activo en el último año, cifra que asciende al 40% si tenemos en cuenta los dos últimos años. El número medio de *commits* al día registrado desde los inicios del proyecto llega casi al millar. Las herramientas de ayuda al desarrollo que utiliza el proyecto GNOME son básicamente las mismas que las que se usan en KDE, por lo que no se incidirá en ellas en este apartado.

Tabla 9-7. Estado actual de GNOME

Página web	http://www.gnome.org
Inicio del proyecto	Septiembre 1997
Licencia	GNU GPL y GNU LGPL
Versión actual de GNOME	2.2 (aunque pronto aparecerá la 2.4)
Líneas de código fuente	9.200.000
Número de ficheros (código, documentación, etc.)	228.000
Estimación de coste	$ 400.000.000
Estimación de tiempo de ejecución	11,08 años (133,02 meses)

ANOTACIONES

Estimación de número medio de desarrolladores	250 aproximadamente
Número de subproyectos	Cuenta con más de 700 módulos en el CVS
Número aproximado de desarrolladores	Casi 1000 con acceso de escritura al CVS
Número de *commiters* activos en el último año	Alrededor de 500 (aproximadamente el 55% del total)
Número de *commiters* activos en los dos últimos años	Unos 700 (75% del total)
Número de *commits* en el CVS	1.900.000
Número medio de *commits* (totales) al día	Unos 900
Herramientas de ayuda al desarrollo	CVS, listas de correo, sitio web, sitio de noticias, reuniones anuales...

Mientras que en KDE, C++ es indiscutiblemente el lenguaje más utilizado, en GNOME el puesto más alto corresponde a C. En GNOME, al igual que en KDE, este hecho se debe a que la biblioteca principal está escrita en C, por lo que el lenguaje *nativo* es ése, mientras que para programar con el resto de lenguajes se ha de esperar a que aparezcan los enlaces. El enlace más avanzado de GNOME es el que está incluido en gnome--, que no es otro que el de C++, por lo que no resulta sorprendente que el segundo lenguaje en la clasificación sea ése. Perl ha tenido desde siempre una amplia aceptación dentro de la comunidad GNOME y siempre se ha puesto como ejemplo de que en GNOME se puede programar en multitud de lenguajes. Su puesta en práctica, sin embargo, no ha sido tan amplia como cabría esperar y supera ligeramente a Shell. Por otro lado, tanto la aceptación de Python como de LISP en GNOME ha sido bastante grande, como se puede desprender de su relativa importancia en esta clasificación, mientras que Java nunca ha llegado a despegar – probablemente debido a un enlace incompleto.

Tabla 9-8. Lenguajes de programación utilizados en GNOME

Lenguaje de programación	Líneas de código	Porcentaje
C	7.918.586	86,10%
C++	576.869	6,27%
Perl	199448	2,17%
Shell	159263	1,73%
Python	137380	1,49%
Lisp	88546	0,96%

ANOTACIONES

9.4.6. Estudios académicos sobre GNOME

Como los estudios más significativos de GNOME en el ámbito académico podemos indicar los siguientes:

- Stefan Koch and Georg Schneider

Lectura complementaria

[Ver bibliografía, apartado 138]

Es uno de los primeros grandes estudios de ingeniería del software en el campo del software libre. Los autores del mismo aprovecharon que los datos del desarrollo suelen estar públicamente accesibles para realizar mediciones de esfuerzo y compararlas con los modelos de estimación de costes, esfuerzos y tiempos clásicos. Uno de los modelos clásicos que se comparó fue el que ha sido utilizado en este capítulo, el modelo COCOMO.

- Daniel Germán

Lectura complementaria

[Ver bibliografía, apartado 101]

Hace un rápido repaso a los objetivos de GNOME, su breve historia, así como el uso de las tecnologías que hace el proyecto GNOME.

9.5. Apache

El servidor HTTP Apache es una de las aplicaciones estrella del mundo del software libre, ya que es el servidor web de mayor implantación según la encuesta que realiza en tiempo real [Netcraft:server-survey-august:03]. Así, en mayo de 1999 el 57% de los servidores web corrían bajo Apache, mientras que en mayo del 2003 el porcentaje ha aumentado hasta el 68%. Apache está disponible para todos los sabores de UNIX (BSD, GNU/Linux, Solaris, etc.), Microsoft Windows y otras plataformas minoritarias.

9.5.1. Historia de Apache

En marzo de 1989 Tim Berners Lee, un científico americano que trabaja en el CERN (Suiza), propone un nueva forma de gestionar la ingente cantidad de información de los proyectos del CERN. Se trata de una red de documentos hiperenlazados (hipertexto tal como Ted

ANOTACIONES

Nelson lo denominó ya en 1965); estaba naciendo el WWW. Hubo que esperar hasta noviembre de 1990, hasta que el primer software WWW vio la luz: en un paquete llamado *WorldWideWeb* se incluía un navegador web de interfaz gráfica y un editor WYSIWYG (*What You See Is What You Get*, 'lo que ve en la pantalla es lo que obtiene como resultado'). Dos años después, la lista de servidores WWW contaba con una treintena de entradas, entre las cuales ya se encontraba el NCSA HTTPd.

La verdadera historia de Apache comienza cuando en marzo de 1995, Rob McCool abandona el NCSA. Apache 0.2 vería la luz el 18 de marzo de 1995, basado en el servidor NCSA httpd 1.3 realizado por el propio Rob McCool durante su estancia en NCSA. Durante esos primeros meses, Apache era una colección de parches aplicados al servidor NCSA, hasta que Robert Thau lanzara Shambhala 0.1, una reimplementación casi completa que ya incluía la API para los módulos que ha resultado ser tan exitosa.

Nota

El nombre del proyecto Apache se debe a la filosofía de desarrollo y de organización. Al igual que la tribu de los apaches, los desarrolladores de Apache decidieron que su forma organizativa debía fundamentarse en los méritos personales de los desarrolladores para con el resto de la comunidad Apache. Se ha extendido, sin embargo, la leyenda de que el nombre Apache en realidad se debe a que en los primeros tiempos no dejaba de ser un servidor NCSA parcheado, en inglés, *a patchy server*.

Habría que esperar a enero de 1996 para poder disfrutar de la primera versión estable de Apache, la Apache 1.0, que incluía la carga de módulos en tiempo de ejecución a modo de pruebas, además de otras funcionalidades interesantes. Los primeros meses de ese año fueron especialmente fructíferos para el proyecto, ya que la versión 1.1 se publicó apenas dos meses después contando con módulos de autentificación contra bases de datos (como MySQL). Desde entonces hasta la actualidad, los hitos más grandes del proyecto han sido la total conformidad con el estándar HTTP 1.1 (incluido en abril de 1997 el Apache 1.2), la inclu-

ANOTACIONES

sión de la plataforma Windows NT (que comenzó ya en julio de 1997 con las versiones en pruebas de Apache 1.3), la unificación de los archivos de configuración en uno solo (habría que esperar a octubre de 1998 ya Apache 1.3.3 para ello) y el lanzamiento, todavía en pruebas, de la siguiente generación de Apache, Apache 2.

En el ínterin, en junio de 1998, IBM decidió que el motor tras su producto WebSphere fuera Apache en lugar de desarrollar un servidor HTTP propio. Esto se vio como un gran espaldarazo por parte del gigante azul al proyecto Apache y al software libre en general, aunque para facilitar este hecho hubiera que cambiar ligeramente la licencia Apache original.

9.5.2. Desarrollo de Apache

El servidor HTTP Apache es el proyecto central dentro de los muchos que gestiona la Apache Software Foundation. El diseño modular de Apache ha permitido que exista una serie de proyectos satélite –algunos incluso más grandes en tamaño que el propio Apache– en torno a Apache. De esta forma, el servidor HTTP Apache contiene el núcleo del sistema con las funcionalidades básicas, mientras las funcionalidades adicionales las aportan los diferentes módulos. Los módulos más conocidos son mod_perl (un intérprete del lenguaje de guión Perl empotrado en el servidor web) y Jakarta (un potente servidor de aplicaciones). En los siguientes párrafos, se va a describir solamente el proceso de desarrollo seguido para el servidor HTTP, sin tener en cuenta los demás módulos, que pueden tener modelos parecidos o no.

El desarrollo del servidor HTTP Apache se fundamenta en el trabajo de un reducido grupo de desarrolladores denominado Apache Group. El Apache Group lo constituyen aquellos desarrolladores que han colaborado durante un periodo prolongado de tiempo, generalmente más de seis meses. Después de ser nominado un miembro del Apache Group para formar parte del mismo, se vota entre todos los miembros del Apache Group. En sus comienzos, el Apache Group constaba de ocho desarrolladores, luego, de doce y en la actualidad cuenta con 25 personas.

ANOTACIONES

Sobre el Apache Group recae la responsabilidad de la evolución del servidor web y, por tanto, las decisiones puntuales de desarrollo en cada momento. Hay que diferenciar el Apache Group del núcleo de desarrolladores (*core group*) activo en cada momento. El carácter voluntario de la mayoría de los desarrolladores hace que sea improbable que todos los que componen el Apache Group puedan estar activos todo el tiempo, por lo que el *core* se define como aquellos que en un espacio de tiempo pueden ocuparse de las tareas en Apache. En líneas generales, las decisiones que han de tomar los desarrolladores pertenecientes al núcleo se limitan a votar la inclusión de código –aunque esto se reserve en realidad sólo para grandes cambios– y para cuestiones de diseño. Por otra parte, en general suelen tener derecho de escritura en el repositorio CVS, por lo que sirven como puerta de entrada del código asegurando que sea correcto, al tiempo que su calidad.

9.5.3. Radiografía de Apache

Las cifras que se exponen a continuación corresponden a la versión del servidor HTTP Apache tal como se podía descargar del servidor CVS del proyecto Apache el 18 de abril del 2003. No se han tenido en cuenta ninguno de los numerosos módulos con los que cuenta el proyecto Apache. Como se puede observar, Apache es un proyecto relativamente pequeño en comparación con los demás casos de estudios considerados en este capítulo. Aunque aquí se ha comentado con anterioridad, es importante hacer hincapié en la modularidad de Apache, que permite precisamente esto: que el núcleo sea pequeño y manejable. El repositorio CVS del proyecto Apache, que contiene el núcleo del servidor web y muchos módulos adicionales, alberga en total más de cuatro millones de líneas de código fuente, una cifra ligeramente inferior a proyectos como KDE y GNOME.

La versión 1.3 de Apache contaba con poco más de 85.000 líneas de código fuente, una cifra que según el modelo COCOMO requeriría un esfuerzo de desarrollo de 20 desarrolladores a tiempo completo de media durante un año y medio. El coste total del proyecto rondaría entonces los cuatro millones de dólares. En la elaboración del servidor web de Apache se cuentan hasta 60 *commiters* diferentes, mientras que el número de desarrolladores que han aportado se calcula que ronda los 400.

ANOTACIONES

Tabla 9-9. Estado actual de Apache

Página web	http://www.apache.org
Inicio del proyecto	1995
Licencia	Apache Free Software License
Versión actual	1.3 (estable), 2.0 (en desarrollo)
Líneas de código fuente	85.000
Número de ficheros	1.206
Estimación de coste	4.000.000 $
Estimación de tiempo de ejecución	1,5 años (18 meses)
Estimación de número medio de desarrolladores	20
Número aproximado de desarrolladores	60 *commiters* (400 desarrolladores)
Herramientas de ayuda al desarrollo	CVS, listas de correo, sistema de notificación de errores

Apache 1.3 está escrito casi íntegramente en el lenguaje C, siendo la presencia de otros lenguajes de programación escasa, sobre todo si tenemos en cuenta que la gran mayoría de las líneas escritas en el segundo lenguaje, Shell, corresponden a ficheros de configuración y de ayuda a la compilación.

Tabla 9-10. Lenguajes de programación utilizados en Apache

Lenguaje de programación	Líneas de código	Porcentaje
C	79015	92%
Shell	5097	7%
Perl	1282	1,7%

9.6. Mozilla

Mozilla es una suite de Internet libre y multiplataforma que incluye un navegador WWW, un cliente de correo electrónico, un cliente de noticias y un editor HTML. En un futuro, sin embargo, Mozilla dejará de ser una aplicación para pasar a ser una plataforma de desarrollo de otras aplicaciones, algo que en parte ya ha venido siendo. Así, muchos otros navegadores utilizan Gecko, el motor de Mozilla, para sus

ANOTACIONES

propósitos, como es el caso de Galeon –un navegador web disponible para GNOME. La siguiente generación de herramientas basadas en Mozilla, como el navegador web Mozilla Firebird, no sólo utilizará el motor Gecko, sino que también hará uso de tecnologías de interfaz de usuario proporcionadas por el proyecto Mozilla.

9.6.1. Historia de Mozilla

La historia de Mozilla es larga y enrevesada, pero al mismo tiempo muy interesante, ya que permite seguir la historia del propio WWW. Y es que si trazamos las personas e instituciones que han estado involucradas en el desarrollo de Mozilla, llegaremos al punto de partida de la web con el lanzamiento del primer navegador web completo.

Al igual que con el antecesor de Apache, fue en el NCSA donde también *nació* el primer navegador web completo en 1993 llamado Mosaic. Muchos de los miembros del equipo de desarrollo, con Marc Andreesen y Jim Clark a la cabeza, crearon una pequeña empresa para escribir, empezando desde cero (ya que había problemas con los derechos de autor del código de Mosaic, y el diseño técnico del programa tenía sus limitaciones, ver [quittner98:_speed_net]), lo que más tarde sería el navegador Netscape Communicator, que fue líder indiscutible del mercado de los navegadores web hasta la llegada del Microsoft Internet Explorer. Además de la innovación puramente tecnológica que supuso el navegador Netscape, Netscape Inc. también fue innovadora en la forma de conseguir copar el mercado. Contra todo el sentido común de la época, su aplicación estrella, el navegador WWW, se podía obtener (y distribuir con ciertas limitaciones) de manera gratuita. Este hecho, hasta entonces insólito dentro del mundo empresarial, causó cierta sorpresa, pero demostró a la larga ser un acierto para la estrategia de Netscape Inc., que sólo un gigante como Microsoft supo atajar con una táctica más agresiva (y probablemente lesiva con la libre competencia).

Para 1997, la cota de mercado de Netscape había caído en picado debido a la implantación de Microsoft Explorer, por lo que desde Netscape Inc. se estudiaban nuevas fórmulas para volver a ganarlo. Un informe técnico del ingeniero Frank Hecker [hecker:setting-up-shop:98] propuso que la mejor solución al problema era liberar el

ANOTACIONES

código fuente del navegador y beneficiarse de los efectos de la comunidad del software libre tal como Eric Raymond describía en "La catedral y el bazar". En enero de 1998 Netscape Inc. anunció oficialmente que liberaría el código fuente de su navegador, marcando un hito de gran importancia dentro de la corta historia del mundo del software libre: una empresa iba a publicar todo el código fuente de una aplicación hasta entonces comercial bajo una licencia de software libre. La fecha indicada para el lanzamiento era el 31 de marzo de 1998.

Lectura complementaria

[Ver bibliografía, apartado 113]

En los dos meses que van de enero a marzo, la actividad en Netscape para que todo estuviera a punto fue frenética. La lista de tareas era enorme y compleja. En el plano técnico, había que contactar con las empresas que habían realizado módulos para pedir su consentimiento en el cambio de licencia; en caso de negativa, el módulo debía ser eliminado. Además, todas las partes escritas en Java debían ser reimplementadas, ya que se consideró que Java no era libre. Se decidió llamar al proyecto libre Mozilla, tal como los propios desarrolladores de Netscape llamaban al componente principal de Netscape y se adquirió el dominio Mozilla.org para construir una comunidad de desarrolladores y colaboradores en torno a ese sitio web. Al final del proceso, se liberó más de un millón y medio de líneas de código fuente.

Nota

El nombre de Mozilla es un juego de palabras con un toque humorístico del equipo de desarrollo en Netscape Inc. Mozilla es el producto de la adaptación de GodZilla, un monstruo que causaba pánico en las películas de terror japonesas desde la década de los cincuenta, para que sonara como *Mosaic Killer*, ya que se pretendía que Mosaic quedara obsoleto gracias a los avances de este nuevo navegador con tecnologías mucho más avanzadas.

Por otra parte, estaba el plano legal. Las licencias libres existentes en aquel momento no convencían a los ejecutivos de Netscape que veían que no *congeniaban* con el carácter comercial de una compañía. Netscape quería una licencia más *flexible* que permitiera llegar a acuerdos con terceros para incluir su código indiferentemente de

ANOTACIONES

su licencia o que otros desarrolladores comerciales pudieran contribuir pudiendo defender sus intereses económicos como desearan. Y aunque en un principio no se había previsto crear una nueva licencia, se llegó a la conclusión de que era la única forma de conseguir lo que deseaban. Así es como se fraguó la Netscape Public License (NPL), una licencia que se basaba en los principios básicos de las licencias de software libre, pero que concedía unos derechos adicionales a Netscape Inc. –lo que la hacía también una licencia *no libre* bajo el prisma de la Free Software Foundation. Cuando se publicó el borrador de la NPL para su discusión pública, llovieron las críticas sobre la cláusula de derechos adicionales para Netscape. Netscape Inc. reaccionó rápidamente ante estos comentarios y creó una licencia adicional, la Mozilla Public Licence (MPL) que era idéntica a la NPL, pero sin que Netscape tuviera ningún derecho adicional.

La decisión final fue liberar el código de Netscape bajo la licencia NPL que otorgaba derechos adicionales a Netscape, y que el código nuevo que fuera incluido estuviera bajo la MPL (o compatible). Las correcciones al código original (licenciado con la NPL), debían estar también bajo esa licencia.

Nota

En la actualidad, Mozilla acepta contribuciones bajo la licencia propia MPL, la GPL y la LGPL. El cambio de licencia no fue nada fácil, ya que hubo que buscar a todos los que habían contribuido con código alguna vez para que dieran su visto bueno al cambio de NPL/MPL a MPL/GPL/LGPL. Con el objeto de poder relicenciar todo el código, se creó una página web que contenía un listado de 300 *desarrolladores perdidos*. A día de hoy, dos años después del cambio de licencia, todavía se buscan unos cuantos desarrolladores.

vEl desarrollo con el código original del Communicator de Netscape fue, sin lugar a dudas, más complicado de lo inicialmente esperado. Las condiciones de partida ya eran malas de por sí, porque lo que fue liberado en ocasiones estaba incompleto (se habían quitado todos los módulos de terceros que no habían dado su visto bueno a la liberación) y corría a duras penas. Por si fuera poco, al problema técnico de pretender que Mozilla corriera sobre una gran cantidad de

Lectura complementaria

[Ver bibliografía, apartado 32]

Lectura complementaria

[Ver bibliografía, apartado 215]

ANOTACIONES

sistemas operativos y plataformas, había que añadir los vicios adquiridos de Netscape Inc., con ciclos de liberación largos e ineficientes para el mundo de Internet y que no distinguía entre sus intereses y los de una comunidad en torno a Mozilla. Todo ello llevó a que, justo un año después, uno de los programadores más activos antes y después de la liberación, Jamie Zawinsky, decidiera tirar la toalla en una amarga carta donde mostraba su desesperación y su desolación.

A pesar de todas las dudas suscitadas y de la existencia de periódicos rumores abocando el proyecto Mozilla al fracaso, varios años después de su liberación, el proyecto goza de buena salud. Su versatilidad y portabilidad ha propiciado que, aun siendo una aplicación muy necesitada de recursos, sea considerada como la pareja de OpenOffice.org a la hora de conquistar el escritorio del usuario final. Con ello, Mozilla ha pasado por varias fases interesantes: comenzó siendo un hito histórico en favor de la implantación y generalización del software libre en sus comienzos, luego padeció una larga "travesía por el desierto" de la que muchos predecían que no iba a salir nunca y por fin ha devenido un proyecto con perspectivas de convertirse en una aplicación de amplio uso. Podemos afirmar con un grado de incertidumbre bastante bajo que si Netscape Inc. no hubiera liberado a principios de 1999 el código fuente de su navegador, hoy sería una aplicación *muerta*.

9.6.2. Estado actual de Mozilla

El 15 de julio del 2003, Netscape Inc. (propiedad de American On-Line) anunció que dejaría de desarrollar el navegador Netscape y, por tanto, la tutela activa del proyecto Mozilla. Como *finiquito*, aprobó la creación de la Fundación Mozilla.org, a la cual apoyó por medio de una aportación de dos millones de dólares. Asimismo, todo el código que se encontraba bajo la NPL (la licencia pública de Netscape) se liberó con las licencias promulgadas ya anteriormente por el proyecto Mozilla: MPL, LGPL y GPL.

Una de las primeras consecuencias de la creación de la Fundación Mozilla.org fue el cambio de estrategia del proyecto Mozilla: si antes primaba la plataforma de desarrollo que posibilitaba la creación de aplicaciones de Internet, en especial web, ahora se pretende que el enfoque principal radique en el usuario final.

ANOTACIONES

9.6.3. Radiografía de Mozilla

Las medidas que se van a presentar en este apartado corresponden al estudio del repositorio CVS del proyecto Mozilla tal como se podía descargar el 22 de agosto del 2003. Los tres millones y medio de líneas de código fuente que encontramos en Mozilla hacen que esta aplicación pueda rivalizar en cuanto a tamaño con el propio núcleo Linux y OpenOffice.org como la aplicación de software libre más grande. Ciertamente, no todo el código que está incluido en el repositorio CVS corresponde a la suite Mozilla, ya que también están incluidas herramientas de ayuda al desarrollo como tinderbox y BugZilla, pero a grandes rasgos, éstas no deberían suponer más allá de un 5% del total de las líneas computadas.

Según las estimaciones del modelo COCOMO, una compañía que quisiera crear un software de tales dimensiones tendría que invertir unos 140 millones de dólares para obtenerlo. El tiempo que habría que esperar se situaría en torno a los siete años y medio y el número medio de programadores a tiempo completo que debería emplear rondaría los 140.

Tabla 9-11. Estado actual de Mozilla

Página web	http://www.mozilla.org
Inicio del proyecto	1998
Licencia	MPL/LGPL/GPL
Versión actual	1.4
Líneas de código fuente	3.500.000
Estimación de coste	$ 140.000.000
Estimación de tiempo de ejecución	7,5 años (90.80 meses)
Estimación de número medio de desarrolladores	140
Número aproximado de desarrolladores	50 *commiters*
Herramientas de ayuda al desarrollo	CVS, listas de correo

En cuanto a los lenguajes de programación, C++ y C son, en este orden, los lenguajes más utilizados. Sorprende, sin embargo, que ninguno de los dos alcance cotas tan altas como las que suelen tener en otras aplicaciones (donde el lenguaje principal suele tener una

ANOTACIONES

cuota cercana al noventa por ciento del total). Ello se debe en gran parte al diseño modular adoptado por Mozilla y que da ciertas libertades en cuanto a la implementación de cada módulo, y entre ellas la posibilidad de elegir el lenguaje que se considere oportuno. Por la misma razón, la presencia de Java –un lenguaje por lo común poco frecuente en el software libre– en el tercer lugar no debería resultar sorprendente, aun teniendo en cuenta que todas esas líneas de código han sido añadidas después de la liberación de Netscape en 1998 que no contenía código en Java.

La presencia de Perl se debe en gran parte a que las herramientas de ayuda al desarrollo realizadas por el proyecto Mozilla como BugZilla o Tinderbox están realizadas en este lenguaje. Lo que sí que resulta algo sorprendente es el alto número de líneas de código en ensamblador en una aplicación de usuario final. La inspección del código en el repositorio ha mostrado que, efectivamente, existen bastantes ficheros codificados en lenguaje ensamblador.

Tabla 9-12. Lenguajes de programación utilizados en Mozilla

Lenguaje de programación	Líneas de código	Porcentaje
C++	1928671	54,61%
C	1156594	32,75%
Java	248802	7,04%
Perl	126534	3,58%
LISP	23165	0,66%
Ensamblador	15493	0,44%

9.7. OpenOffice.org

OpenOffice.org es una de las aplicaciones estrella del panorama actual del software libre. Se trata de una suite de oficina multiplataforma que incluye las aplicaciones clave en un entorno de escritorio de oficina, tales como son el procesador de texto (Writer), hoja de cálculo (Calc), gestor de presentaciones (Impress), un programa de dibujo (Draw), un editor de fórmulas matemáticas (Math) y finalmente un editor de lenguaje HTML (incluido en Writer). El interfaz que ofrece OpenOffice.org es homogéneo e intuitivo, similar en aspecto y fun-

cionalidades a otras suites de oficina, en especial a la suite más arraigada en la actualidad, Microsoft Office.

Escrito en C++, OpenOffice.org incluye la API de Java y tiene su propio sistema de componentes empotrables, que permite incluir, por ejemplo, tablas de la hoja de cálculo en el procesador de textos de una manera sencilla e intuitiva. Entre sus ventajas está que maneja una gran cantidad de formatos de archivo, incluidos los de Microsoft Office. Sus formatos de archivo nativos, al contrario que los de la suite de Microsoft, están basados en XML, por lo que se demuestra una clara apuesta por la versatilidad, la facilidad de transformación y la transparencia. En la actualidad, OpenOffice.org está traducido a más de 25 lenguas y corre en Solaris (su sistema nativo), GNU/Linux y Windows. En un futuro no muy lejano se esperan versiones para FreeBSD, IRIX y MacOS X.

OpenOffice.org adoptó su nombre definitivo (a OpenOffice –tal como la conoce todo el mundo– más la coletilla *.org*) después de un litigio en el que fue demandado por usurpación de nombre de marca por otra empresa.

9.7.1. Historia de OpenOffice.org

A mediados de la década de los ochenta, se fundó en la República Federal de Alemania la empresa StarDivision, que tuvo como objetivo principal la creación de una suite ofimática: StarOffice. En verano de 1999, SUN Microsystems –con la clara intención de arrebatarle un mercado conquistado ya por aquel entonces a Microsoft– decidió adquirir la empresa StarDivision y hacer una fuerte apuesta por StarOffice. De este modo, en junio del 2000 lanzó la versión 5.2 de StarOffice que podía ser descargada de manera gratuita a través de la Red.

Sin embargo, el éxito de StarOffice fue limitado, ya que el mercado estaba fuertemente dominado por la suite de Microsoft. SUN decidió cambiar su estrategia y, al igual que Netscape con el proyecto Mozilla, decidió aprovechar las ventajas del software libre para ganar en importancia e implantación. Así, las futuras versiones de StarOffice (producto propietario de SUN) serían creadas utilizando OpenOffice.org (producto libre) como fuente, respetando las interfaces de programación (API), los formatos de fichero y sirviendo como implementación de referencia.

ANOTACIONES

9.7.2. Organización de OpenOffice.org

OpenOffice.org pretende tener una estructura decisoria donde todos los miembros de la comunidad se sientan partícipes. Por eso, se ha ideado un sistema para que la toma de decisiones cuente con el mayor consenso posible. El proyecto OpenOffice.org se divide en una serie de subproyectos que cuentan con unos miembros del proyecto, los colaboradores, y un único líder. Por supuesto, los miembros de un proyecto pueden participar en más de uno, al igual que el líder. Sin embargo, sólo se puede liderar un proyecto a la vez. Los proyectos, a su vez, se dividen en tres categorías:

- Proyectos aceptados: pueden ser tanto de carácter técnico como no técnicos. Los líderes de cada proyecto aceptado cuentan con un voto a la hora de la toma de decisiones globales.

- Proyectos *Native-lang*: engloba todos los proyectos de internacionalización y localización de OpenOffice.org. En la actualidad, como se ha comentado con anterioridad, existen más de 25 equipos trabajando para traducir las aplicaciones de OpenOffice.org a las diferentes lenguas y convenciones. En conjunto, native-lang cuenta con un único voto para la toma de decisiones globales.

- Proyectos en la incubadora: se trata de proyectos patrocinados por la comunidad (generalmente experimentales o pequeños). Pueden pasar a ser aceptados tras un periodo de seis meses. De este modo, la comunidad OpenOffice.org puede garantizar que los proyectos aceptados están respaldados por un interés real, ya que la *mortalidad* de proyectos nuevos en el mundo del software libre es muy grande. En total, los proyectos en la incubadora cuentan con un voto en la toma de decisiones.

9.7.3. Radiografía de OpenOffice.org

La suite ofimática OpenOffice.org está compuesta de cerca de cuatro millones de líneas de código fuente, distribuidas a lo largo de 45 mil ficheros.

ANOTACIONES

El modelo COCOMO estima el esfuerzo necesario para realizar un clon de OpenOffice.org en 150 programadores trabajando durante casi ocho años a tiempo completo. El coste de desarrollo ascendería, según las estimaciones de COCOMO, a unos 160 millones de dólares.

Los resultados que se comentan en este apartado fueron obtenidos del estudio del repositorio CVS del proyecto OpenOffice.org el 20 de abril del 2003.

Tabla 9-13. Estado actual de OpenOffice.org

Página web	http://www.OpenOffice.org
Inicio del proyecto	Junio del 2000 (Primera versión libre)
Licencia	LGPL y SISSL
Versión actual	1.0.3 (en desarrollo la 1.1)
Líneas de código fuente	4.000.000
Número de ficheros	45.000
Estimación de coste	$ 160.000.000
Estimación de tiempo de ejecución	8 años (95 meses)
Estimación de número medio de desarrolladores	150
Número aproximado de desarrolladores	200 *commiters*
Herramientas de ayuda al desarrollo	CVS, listas de correo

En cuanto a los lenguajes de programación utilizados en OpenOffice.org, el primer lugar lo ocupa C++. Es interesante observar cómo la adquisición por parte de SUN implica la integración de mucho código Java en la suite ofimática, superando incluso a C. Del resto de lenguajes más utilizados, Pascal debe obtener una mención especial, ya que suele ser un lenguaje poco utilizado en el mundo del software libre y, en general, podríamos decir en la producción industrial de software.

Tabla 9-14. Lenguajes de programación utilizados en OpenOffice.org

Lenguaje de programación	Líneas de código	Porcentaje
C++	3361765	85,03%
Java	251191	6,35%

ANOTACIONES

Lenguaje de programación	Líneas de código	Porcentaje
C	222533	5,63%
Pascal	32407	0,82%
Perl	28380	0,72%

9.8. Red Hat Linux

Red Hat Linux fue una de las primeras distribuciones comerciales de GNU/Linux. Hoy en día es probablemente una de las más conocidas, y seguramente la que se puede considerar como la "canónica" de entre las distribuciones comerciales. El trabajo de los distribuidores está básicamente relacionado con tareas de integración y no tanto con el desarrollo de software. Por supuesto, tanto Red Hat como otras distribuciones pueden contar con desarrolladores entre sus empleados, pero su labor es secundaria para los objetivos de una distribución. En general, se asume que el trabajo que realizan las distribuciones es simplemente el de tomar los paquetes fuente (generalmente los archivos que publican los propios desarrolladores) y empaquetarlos de forma que cumplan ciertos criterios (tanto técnicos como organizativos). El producto de este proceso es una distribución: una serie de paquetes convenientemente organizados que posibilitan al usuario instalarlos, desinstalarlos y actualizarlos.

Las distribuciones también son responsables de la calidad del producto final, un aspecto muy importante teniendo en cuenta que muchas de las aplicaciones que incluyen han sido elaboradas por voluntarios en su tiempo libre. Aspectos de seguridad y estabilidad son, por tanto, de capital importancia para una distribución.

9.8.1. Historia de Red Hat

Lectura complementaria

[Ver bibliografía, apartado 180]

Red Hat Software Inc. fue fundada en 1994 por Bob Young y Marc Ewing. Su principal objetivo era compilar y comercializar una distribución GNU/Linux, que vino a llamarse (y todavía se sigue llamando) Red Hat Linux [young:giving-it-away:99]. Básicamente, se trataba de una versión empaquetada de lo que existía en la Red en aquellos tiempos, incluyendo documentación y soporte. Du-

rante el verano de 1995, la versión 1.0 de esta distribución vio la luz. Unos meses más tarde, en otoño, se publicó la versión 2.0, que incluía la tecnología RPM (RPM *Package Manager*, gestor de paquetes RPM). El sistema de paquetes RPM se ha convertido en un estándar *de facto* para los paquetes de sistemas GNU/Linux. 1998 fue el año en el que Red Hat llegó al gran público con la versión 5.2. Para una historia completa de los nombres de las diferentes versiones de Red Hat, véase Stephen J. Smoogen. *The truth behind red hat names*.

Nota

Desde la versión 1.1 del Linux Standard base (una especificación cuya meta es conseguir compatibilidad binaria entre las distribuciones GNU/Linux de la que se encarga el Free Standards Group), RPM ha sido elegido como el sistema de paquetes estándar. El proyecto Debian continúa con su formato de paquete propio, así como muchas de las distribuciones dependientes del sistema de paquetes Debian, y se ajustan al formato estandarizado mediante una herramienta de conversión que se llama *alien*.

Antes de la existencia del sistema de gestión de paquetes RPM, casi todas las distribuciones de GNU/Linux ofrecían la posibilidad de instalar el software mediante un procedimiento dirigido por menús, pero hacer modificaciones a una instalación ya hecha, especialmente la agregación de paquetes de software nuevos tras la instalación, no era una tarea fácil. RPM permitió dar ese paso más allá del estado del arte al proveer a los usuarios con la posibilidad de gestionar sus paquetes, lo que permitiría borrar, instalar o actualizar cualquier paquete software existente en la distribución de manera mucho más sencilla. El sistema de paquetes RPM sigue siendo el sistema de paquetes más utilizado entre las diferentes distribuciones de GNU/Linux. Las estadísticas de Ladislav Bodnar (2003). *Linux distributions - facts and figures*, un sitio web que contiene información cualitativa y cuantitativa acerca de un gran número de distribuciones, muestran que en mayo del 2003 una gran mayoría de las 118 distribuciones computadas utilizan el gestor RPM, en total 65 (lo que viene a ser un 55% del total). En comparación, el sistema de paquetes de Debian (conocido como deb) lo utiliza únicamente 16 distribuciones (un 14% del total).

Lectura complementaria

[Ver bibliografía, apartados 65 y 73]

ANOTACIONES

Sin embargo, Red Hat Inc. no sólo es conocida por su distribución de software basada en Linux. En agosto de 1999, Red Hat salió a bolsa y sus acciones obtuvieron la octava ganancia de primer día más grande en toda la historia de Wall Street. Cuatro años más tarde, el valor de las acciones de Red Hat es en torno a una centésima parte del máximo valor que llegara a alcanzar antes de la crisis de las puntocom. Aun así, sus comienzos exitosos en el mercado de valores sirvieron para que Red Hat fuera portada en periódicos y revistas no directamente relacionadas con temas informáticos. En cualquier caso, parece ser que Red Hat ha sabido superar los problemas de otras compañías del mundo de los negocios en torno al software libre y anunció números negros por primera vez en su historia en el último cuarto del año 2002.

Lectura complementaria

[Ver bibliografía, apartado 194]

Otro de los hechos históricos más importantes de Red Hat fue la adquisición en noviembre de 1999 de Cygnus Solutions, una empresa fundada una década antes y que ya había demostrado cómo con una estrategia integral basada en software libre se puede ganar dinero. Cygnus escogió el exigente mercado de los compiladores para hacerse un hueco. Su estrategia comercial se basaba en el desarrollo y la adaptación de las herramientas de desarrollo de software GNU (básicamente GCC y GDB) a petición del cliente.

9.8.2. Estado actual de Red Hat

En la actualidad, los productos estrella de Red Hat Inc. son Red Hat Linux 9 y Red Hat Network, un servicio de actualización de software a través de la red. Este tipo de servicios están más bien orientados al usuario final y no tanto al entorno empresarial, pero sirven a Red Hat como buen reclamo y para asegurar su estrategia de marca.

La *verdadera* estrategia comercial de Red Hat se encuentra en sus productos dirigidos al mundo empresarial. Este tipo de productos son mucho menos conocidos, pero suponen una gran parte de la facturación de Red Hat, muy superior a la que percibe por sus productos *estrella*, más populares en el sentido literal.

Red Hat cuenta con una distribución orientada a la empresa, integrada en torno a un servidor de aplicaciones y llamada Red Hat Enterprise Linux AS. Con la adquisición de este software, el cliente tiene

ANOTACIONES

derecho a soporte. El servicio análogo a Red Hat Network para usuarios comerciales es Red Hat Enterprise Network, que incluye la gestión del sistema y la posibilidad de actualizaciones. Por otro lado, Red Hat ofrece también servicios de consultoría informática y un programa de certificación similar al que existe en el mundo Windows y ofrecido por Microsoft.

9.8.3. Radiografía de Red Hat

Red Hat ha superado recientemente la barrera de los 50 millones de líneas de código que hacen de ella una de las mayores distribuciones de software que han llegado a existir – superando como se verá más adelante en este capítulo el tamaño de sistemas operativos propietarios. La versión 8.1 de Red Hat, inmediatamente anterior a la última, estaba constituida por 792 paquetes, por lo que podemos asumir que también habrá franqueado el listón de los 800 paquetes en su última versión, teniendo en cuenta que su número suele incrementarse levemente de versión en versión.

Al igual que en los casos anteriores, se ha aplicado el modelo COCOMO para estimar la inversión y el esfuerzo que sería necesario emplear en la generación de un software de idéntico tamaño. Sin embargo, para el caso de Red Hat se ha tenido en cuenta que se trata de un producto realizado a partir de una serie de aplicaciones independientes. Por eso, se ha realizado una estimación mediante COCOMO, independiente para cada uno de los paquetes de Red Hat para luego sumar el coste y personal total necesario. En el caso del tiempo de ejecución óptimo para Red Hat se ha considerado el del paquete más grande, ya que idealmente, al ser todos los paquetes independientes, podrían realizarse de manera paralela en el tiempo. Es por ello, por lo que el tiempo de ejecución óptimo para Red Hat es parecido al de los proyectos que se han presentado con anterioridad en este capítulo.

Según COCOMO, se necesitarían alrededor de siete años y medio y un equipo de programadores compuesto por una media de 1.800 desarrolladores para realizar la distribución Red Hat Linux 8.1 desde cero. El coste de desarrollo final ascendería a unos 1.800 millones de dólares.

ANOTACIONES

ANOTACIONES

Nota

1.800 millones de dólares es el presupuesto que el Ministerio de Defensa español destinará a renovar su flota de helicópteros. De todo el montante, la mitad irá destinado a la adquisición de 24 helicópteros, por lo que el precio de Red Hat sería el equivalente a 48 helicópteros de combate. Asimismo, la cifra de 1.800 millones de dólares es la que obtuvo como beneficios a nivel mundial la película *Titanic*.

Tabla 9-15. Estado actual de Red Hat Linux

Página web	http://www.redhat.com
Inicio del proyecto	1993
Licencia	
Versión actual	9.0
Líneas de código fuente	Más de 50.000.000
Número de paquetes	792
Estimación de coste	$ 1,800,000,000
Estimación de tiempo de ejecución	7,35 años (88.25 meses)
Estimación de número medio de desarrolladores	1.800
Número aproximado de desarrolladores	Empleados de Red Hat (generalmente sólo integración)
Herramientas de ayuda al desarrollo	CVS, listas de correo

Debido a la presencia de un amplio número de paquetes, la clasificación de lenguajes en Red Hat cuenta con mayor diversidad que las que hemos visto en las aplicaciones más importantes de software libre. En términos generales, se percibe la gran importancia de C con más de un sesenta por ciento de las líneas de código. En segundo lugar, con más de 10 millones de líneas de código se encuentra C++, seguido de lejos por Shell. Es interesante observar que a Perl se unen después LISP (mayoritariamente debido a su uso en Emacs), el código en ensamblador (del cual una cuarta parte corresponde al que viene con Linux) y un lenguaje en franco retroceso en cuanto a su uso, como es Fortran.

Tabla 9-16. Lenguajes de programación utilizados en Red Hat

Lenguaje de programación	Líneas de código	Porcentaje
C	30993778	62,13%
C++	10216270	20,48%
Shell	3251493	6,52%
Perl	1106082	2,22%
LISP	958037	1,92%
Ensamblador	641350	1,29%
Fortran	532629	1,07%

9.9. Debian GNU/Linux

Debian es un sistema operativo libre que en la actualidad utiliza el núcleo de Linux para llevar a cabo su distribución (aunque se espera que existan distribuciones Debian basadas en otros núcleos, como es el caso con The HURD, en el futuro). Actualmente está disponible para varias arquitecturas diferentes, incluyendo Intel x86, ARM, Motorola, 680x0, PowerPC, Alpha y SPARC.

Debian no es sólo la distribución GNU/Linux más grande en la actualidad, también es una de las más estables y disfruta de varios premios en cuanto a la preferencia de los usuarios. Aunque su base de usuarios sea difícil de estimar, ya que el proyecto Debian no vende CD u otros medios con su software, y el software que contiene puede ser redistribuido por cualquiera que así lo desee, podemos suponer sin faltar mucho a la verdad que se trata de una distribución importante dentro del mercado de GNU/Linux.

En Debian existe una categorización según la licencia y los requisitos de distribución de los paquetes. El núcleo de la distribución Debian (la sección llamada *main,* que aglutina una gran variedad de paquetes) está compuesto sólo por software libre de acuerdo con las (Debian Free Software Guidelines). Está disponible en Internet para ser descargado y muchos redistribuidores lo venden en CD u otros medios.

Lectura complementaria

[Ver bibliografía, apartado 85]

ANOTACIONES

Las distribuciones de Debian son creadas por cerca de un millar de voluntarios (generalmente profesionales de la informática). La labor de estos voluntarios radica en tomar los programas fuente –en la mayoría de los casos de sus autores originales–, configurarlos, compilarlos y empaquetarlos, de manera que un usuario típico de una distribución Debian sólo tenga que seleccionar el paquete para que el sistema lo añada sin mayores problemas. Esto que a simple vista puede parecer simple, se torna complejo en cuanto se introducen factores como las dependencias entre los diferentes paquetes (el paquete A necesita, para poder funcionar, del paquete B) y las diferentes versiones de todos estos paquetes.

La labor de los integrantes del proyecto Debian es la misma que la que se realiza en cualquier otra distribución: la integración de software para su correcto funcionamiento conjunto. Además del trabajo de adaptación y empaquetamiento, los desarrolladores Debian se encargan de mantener una infraestructura de servicios basados en Internet (sitio web, archivos en línea, sistema de gestión de errores, listas de correo de ayuda, soporte y desarrollo, etc.), de varios proyectos de traducción e internacionalización, del desarrollo de varias herramientas específicas de Debian y, en general, de cualquier elemento que hace que la distribución Debian sea posible.

Lectura complementaria

[Ver bibliografía, apartado 87]

Aparte de su naturaleza voluntaria, el proyecto Debian tiene una característica que lo hace especialmente singular: el contrato social de Debian. Este documento contiene no sólo los objetivos principales del proyecto Debian, sino también los medios que se utilizarán para llevarlos a cabo.

Lectura complementaria

[Ver bibliografía, apartado 86]

Debian también es conocida por tener una política de paquetes y de versionado muy estricta con el fin de conseguir una mayor calidad del producto. Así, en todo momento existen tres *sabores* diferentes de Debian: una versión estable, una inestable y otra en pruebas. Como su propio nombre indica, la versión estable es la versión indicada para sistemas y personas no aptas para sobresaltos. Su software ha de pasar un periodo de congelación en el que sólo se corrigen erratas. La norma es que en la versión estable de Debian no ha de haber ningún error crítico conocido. Por contra, la versión estable de Debian no suele tener las últimas versiones del software (lo más novedoso).

Para los que deseen tener una versión con el software más actual existen otras dos versiones de Debian coetáneas con la estable. La versión inestable incluye paquetes en vía de estabilización, mientras que la versión en pruebas, como su propio nombre indica, es la más proclive a fallar y contiene lo último de lo último en lo que a novedades de software se refiere.

En el momento de este estudio, la versión estable de Debian es Debian 3.0 (también conocida como *Woody*), la inestable recibe el sobrenombre de *Sid* y la que se encuentra en pruebas es *Sarge*. Pero en el pasado, Woody pasó también por una etapa inestable y, antes de eso, otra en pruebas. Esto es importante, porque lo que vamos a considerar en este artículo son las diferentes versiones estables de Debian, desde que se publicara la versión 2.0 allá por 1998. Así, tenemos a Debian 2.0 (alias *Hamm*), Debian 2.1 (*Slink*), Debian 2.2 (*Potato*) y, por último, Debian 3.0 (*Woody*).

Nota

Los apodos de las versiones de Debian corresponden a los protagonistas de la película de dibujos animados *Toy Story*, una tradición que se implantó medio en serio, medio en broma cuando se publicó la versión 2.0 y Bruce Perens, entonces líder del proyecto y después fundador de la Open Source Initiative y del término Open Source, trabajaba para la empresa que se encargaba de realizar esta película. Se pueden encontrar más detalles sobre la historia de Debian y la distribución Debian en general en Bdale Garbee; Hartmut Koptein; Nils Lohner; Will Lowe; Bill Mitchell; Ian Murdock; Martin Schulze; Craig Small. *A brief history of debian*. En el paquete debian-history.

Lectura complementaria

[Ver bibliografía, apartado 100]

9.9.1. Radiografía de Debian

Al igual que en el caso de Red Hat, el producto software final que ofrece Debian ofrece resultados astronómicos. La última versión estable de Debian, la versión 3.0 que recibió el sobrenombre de *Woody*, cuenta con nada menos que 4.579 paquetes fuente y alrededor de 10.000 paquetes binarios. Podemos considerar, a la vista de es-

ANOTACIONES

tos resultados, que Debian es la colección de software integrado más grande que existe y que es probable que debido a su tamaño se tenga que enfrentar en un futuro no muy lejano a posibles problemas de integración.

El número de líneas de código en Debian 3.0 asciende a 105 millones. Según el modelo COCOMO habría que desembolsar una cantidad aproximada de 3.600 millones de dólares para obtener un software como el que viene empaquetado con esta distribución. Al igual que en el caso de Red Hat, se ha computado por separado el esfuerzo necesario para cada paquete y luego se han sumado todas las cantidades. Por la misma razón, el tiempo de desarrollo de Debian asciende sólo a 7 años, ya que se considera que cada paquete puede realizarse paralelamente a los demás en lo que respecta al tiempo. Eso sí, de media habría que movilizar a cerca de cuatro mil desarrolladores durante esos siete años para lograrlo.

Nota

3.600 millones de dólares es el presupuesto que tiene asignado el VI Programa Marco de la Comisión Europea para actividades de investigación y desarrollo relacionadas con la "Sociedad de la Información". También es la cifra que Telefónica prevé invertir en Alemania para implantar UMTS.

Tabla 9-17. Estado actual de Debian

Página web	http://www.debian.org
Inicio del proyecto	16.8.1993
Licencia	Las que cumplan las DFSG
Versión actual de Debian	Debian 3.0 (alias Woody)
Líneas de código fuente	105.000.000
Número de paquetes	4.579
Estimación de coste	3.625.000.000 $
Estimación de tiempo de ejecución	7 años
Estimación de número medio de desarrolladores	Cerca de 3.950
Número aproximado de desarrolladores (*maintainers*)	Cerca de mil
Herramientas de ayuda al desarrollo	Listas de correo, sistema de notificación de errores

El lenguaje de programación más utilizado en Debian 3.0 es C, con más de un 60% de las líneas de código. Sin embargo, como se mostrará un poco más adelante en este apartado, la importancia de C va remitiendo con el tiempo, ya que las primeras versiones de Debian contaban hasta con un 80% del código en C. Buena parte de la *culpa* del retroceso de C lo tiene el segundo en cuestión C++, pero sobre todo la irrupción de los lenguajes de guión (*scripting*) como Perl, Python y –aunque fuera de esta tabla– PHP. En medio se cuelan lenguajes como LISP o algo más exótico, Fortran.

Tabla 9-18. Lenguajes de programación utilizados en Debian GNU/Linux

Lenguaje de programación	Líneas de código	Porcentaje
C	66.549.696	63,08%
C++	13.066.952	12,39%
Shell	8.635.781	8,19%
LISP	4.087.269	3,87%
Perl	3.199.436	3,03%
Fortran	1,939,027	1,84%
Python	1.458.783	1,38%

En la tabla 9-19 se muestra la evolución de los lenguajes más significativos –los que superan el 1% de código– en Debian 3.0. Por debajo de la frontera del 1% se sitúan en Debian 3.0, en este orden, PHP, Ada, Modula3, Objective C, Java, Yacc y ML (todos con porcentajes entre el 0.30% y el 0.60%).

Tabla 9-19. Lenguajes más utilizados en Debian

Lenguaje	Debian 2.0		Debian 2.1		Debian 2.2		Debian 3.0	
C	19.400.000	76,67%	27.800.00	74,89%	40.900.000	69,12%	66.500.000	63,08%
C++	1.600.000	6,16%	2.800.000	7,57%	5.980.000	10,11%	13.000.000	12,39%
Shell	645.000	2,55%	1.150.000	3,10%	2.710.000	4,59%	8.635.000	8,19%
Lisp	1.425.000	5,64%	1.890.000	5,10%	3.200.000	5,41%	4.090.000	3,87%
Perl	425.000	1,68%	774.000	2,09%	1.395.000	2,36%	3.199.000	3,03%
Fortran	494.000	1,96%	735.000	1,98%	1.182.000	1,99%	1.939.000	1,84%
Python	122.000	0,48%	211.000	0,57%	349.000	0,59%	1.459.000	1,38%
Tcl	311.000	1,23%	458.000	1,24%	557.000	0,94%	1.081.000	1,02%

Existen lenguajes a los que podríamos considerar minoritarios que alcanzan un puesto bastante alto en la clasificación. Esto se debe a que aun encontrándose presentes en un número reducido de paquetes, éstos son bastante grandes. Tal es el caso de Ada, que en tres paquetes (gnat, un compilador de Ada, libgtkada, un enlace a la biblioteca GTK, y Asis, un sistema para gestionar fuentes en Ada) aglutina 430.000 de un total de 576.000 líneas de código fuente que se han contabilizado en Debian 3.0 para Ada. Otro caso parecido es el de Lisp, que cuenta sólo con GNU Emacs y con Xemacs, con más de 1.200.000 líneas de los aproximadamente 4 millones de toda la distribución.

9.9.2. Comparación con otros sistemas operativos

Si dice que todas las comparaciones son odiosas, las de software libre con software propietario lo son más. Las radiografías tan detalladas de Red Hat Linux y Debian han sido posibles por su condición de software libre. El acceso al código (y a otra información que ha sido expuesta en este capítulo) es indispensable para estudiar a fondo las diferentes versiones en cuanto a número de líneas, paquetes, lenguajes de programación utilizados, etc. Pero las ventajas del software libre van más allá, porque además facilitan la revisión de terceras personas, ya sean grupos de investigación o sencillamente personas interesadas.

En los sistemas propietarios, en general, realizar un estudio así es tarea imposible. De hecho, las cuentas que se ofrecen a continuación tienen sus fuentes en las propias compañías que están detrás del desarrollo de software, por lo que no podemos avalar su veracidad. Para más inri, en muchos casos no sabemos si se está hablando de líneas de código fuente físicas tal como hemos venido haciendo a lo largo de este capítulo o también incluyen en sus cuentas las líneas en blanco y las de comentarios. A ello hay que añadir que tampoco sabemos a ciencia cierta lo que consideran en su software, por lo que para algunas versiones de Microsoft Windows no sabemos si incluyen el paquete de Microsoft Office o no.

En cualquier caso, y teniendo en cuenta todo lo que se ha comentado al respecto en párrafos anteriores, pensamos que incluir esta comparativa es interesante, ya que nos ayuda a situar las diferentes dis-

ANOTACIONES

tribuciones de Red Hat y de Debian dentro de un panorama más amplio. Lo que parece estar fuera de toda duda es que tanto Debian como Red Hat, pero especialmente el primero, son las colecciones de software más grandes vistas jamás por la humanidad hasta el momento.

Los números que se citan a continuación proceden de Mark Lucovsky (2000). *From nt os/2 to windows 2000 and beyond - a software-engineering odyssey* para Windows 2000, SUN Microsystems (2000). *Sun microsystems announces availability of StarOffice(TM) source code on OpenOffi-ce. org* para StarOffice 5.2, Gary McGraw. "Building secure software: How to avoid security problems the right way". En: David A. Wheeler para Windows XP y Bruce Schneier (2000). *Software complexity and security* para el resto de sistemas. En la tabla tabla 9-20 se muestra la comparativa en orden creciente.

Lectura complementaria

[Ver bibliografía, apartados 147, 150, 148 y 179]

Tabla 9-20. Comparación con sistemas propietarios

Sistema	Fecha de publicación	Líneas de código (aprox.)
Microsoft Windows 3.1	Abril 1992	3.000.000
SUN Solaris 7	Octubre 1998	7.500.000
SUN StarOffice 5.2	2000.06	7.600.000
Microsoft Windows 95	Agosto 1995	15.000.000
Red Hat Linux 6.2	Marzo 2000	18.000.000
Debian 2.0	Julio 1998	25.000.000
Microsoft Windows 2000	Febrero 2000	29.000.000
Red Hat Linux 7.1	Abril 2001	32.000.000
Debian 2.1	Marzo 1999	37.000.000
Windows NT 4.0	Julio 1996	40.000.000
Red Hat Linux 8.0	Septiembre 2002	50.000.000
Debian 2.2	Agosto 2000	55.000.000
Debian 3.0	Julio 2002	105.000.000

ANOTACIONES

Bibliografía

[1] *About issues.* http://www.openoffice.org/project_issues.html

[2] *Ada core technologies.* http://www.gnat.com

[3] *Aegis.* http://aegis.sourceforge.net/index.html

[4] *Alcove.* http://www.alcove.com

[5] *Alcove-labs.* http://www.alcove-labs.org

[6] *Alioth.* http://alioth.debian.org

[7] *The apache ant project.* http://ant.apache.org

[8] *Arch revision control system.* http://arch.fifthvision.net/bin/view

[9] *artofcode LLC.* http://artofcode.com

[10] *Autoconf.* http://www.gnu.org/software/autoconf

[11] *Barrapunto.* http://barrapunto.com

[12] *Berlios: The open source mediator.* http://berlios.de

[13] *Bitkeeper source management.* http://www.bitmover.com/bitkeeper

[14] *Bruce perens.* http://perens.org/OpenStandards

[15] *Caldera.* http://www.sco.com

[16] *Cisco enterprise print system.* http://ceps.sourceforge.net

[17] *Concurrent version system.* http://www.cvshome.org

[18] *Creative commons.* http://creativecommons.org

[19] *Directory of open access journals.* http://www.doaj.org

ANOTACIONES

[20] *ecos*. http://sources.redhat.com/ecos

[21] *ecos license 2.0*. http://www.gnu.org/licenses/ecos-license.html

[22] *First monday: Peer rewiewed journal on the internet*. http://www.first-monday.dk

[23] *Free Software Foundation*. http://www.fsf.org

[24] *Fundación wu ming*. http://www.wumingfoundation.com

[25] *G-forge*. http://gforge.org

[26] *Gettext*. http://www.gnu.org/software/gettext

[27] *GNU automake*. http://www.gnu.org/software/automake

[28] *GNU libc*. http://www.gnu.org/software/libc

[29] *GNU libtool*. http://www.gnu.org/software/libtol

[30] *GNU make*. http://www.gnu.org/software/make/make.html

[31] *GNU troff*. http://www.gnu.org/software/groff/groff.html

[32] *Have you seen these hackers?* http://www.mozilla.org/MPL/missing.html

[33] *History of tex*. http://www.math.utah.edu/software/plot79/tex/history.html

[34] *IBM public license version 1.0*. http://oss.software.ibm.com/developerworks/opensource/license10.html

[35] *Jam product information*. http://www.perforce.com/jam/jam.html

[36] *The Linux documentation project*. http://www.tldp.org

[37] *LinuxCare*. http://www.linuxcare.com

[38] *Mailman, the GNU mailing list manager*. http://www.list.org

[39] *Mozilla public license 1.1*. http://www.mozilla.org/MPL/MPL-1.1.html

[40] *Mozilla tinderbox*. http://www.mozilla.org/tinderbox.html

ANOTACIONES

[41] *Open cores*. http://www.opencores.org

[42] *Open directory project social contract*

[43] *Open Source Initiative*. http://www.opensource.org

[44] *Public development cluster*. http://handhelds.org/projects/skiffcluster.html

[45] *Public library of science*. http://www.publiclibraryofscience.org

[46] *Red hat*. http://www.redhat.com

[47] *Savannah*. http://savannah.gnu.org y http://savannah.nongnu.org

[48] *Slashdot: news for nerds*. http://slashdot.org

[49] *Sleepycat license*. http://www.sleepycat.com/download/oslicense.html

[50] *Sleepycat software*. http://www.sleepycat.com

[51] *Sourceforge: Open source software development website*. http://sourceforge.net

[52] *Subversion*. http://subversion.tigris.org

[53] *Texinfo - the GNU documentation system*

[54] *Tigris.org: Open source software engineering*. http://tigris.org

[55] *W3c document license*. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

[56] *Wikipedia*. http://www.wikipedia.org

[57] *X window system release 11 license*. http://www.x.org/Downloads_terms.html

[58] *Ximian*. http://ximian.com/

[59] *Zope corporation*. http://www.zope.com/

[60] *Zope public license 2.0*. http://www.zope.org/Resources/ZPL

ANOTACIONES

[61] *Ley de Propiedad Intelectual*. Real Decreto Legislativo 1/1996, de 12 de abril, abril 1996

[62] *Affero general public license* (2002). http://www.affero.org/oagpl.html

[63] ISO JTC 1/SC 34 (1986). *Standard generalized markup language (SGML, iso 8879).*

[64] I. Antoniades; I. Samoladas; I. Stamelos; G.L. Bleris. *Dynamical simulation models of the open source development process*. En: Koch. http://wwwai.wu-wien.ac.at/˜koch/oss-book

[65] Edward C. Bailey (1998). *Maximum rpm - taking the red hat package manager to the limit*. http://rikers.org/rpmbook/

[66] Jesús M. González Barahona (2000). "Software libre, monopolios y otras yerbas". *Todo Linux* (núm. 3). http://sinetgy.org/˜jgb/articulos/soft-libre-monopolios

[67] Jesús M. González Barahona (2002). "¿Qué se hace con mi dinero?" *Todo Linux* (vol. 17). http://sinetgy.org/˜jgb/articulos/sobre-administracion

[68] Jesús M. González Barahona; Gregorio Robles. *Libre software engineering web site*. http://libresoft.dat.escet.urjc.es

[69] Jesús M. González Barahona; Gregorio Robles (mayo, 2003). "Unmounting the "code god" assumption". En: *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Enginee-ring.* Italia: Génova.

[70] Jesús M. González Barahona; Gregorio Robles; Miguel A. Ortuño Pérez; Luis Rodero Merino; José Centeno González; Vicente Matellán Olivera; Eva M. Castro Barbero; Pedro de las Heras Quirós. "Anatomy of two GNU/Linux distributions". En: Koch. http://wwwai.wu-wien.ac.at/˜koch/oss-book

[71] Matthew P. Barnson. *The bugzilla guide*. http://www.bugzilla.org/docs214/html/index.html

[72] Nikolai Bezroukov (diciembre, 1998). "A second look at the cathedral and the bazaar". *First Monday* (vol. 12, núm.4). http://www.firstmonday.dk/issues/issue4_12/bezroukov/index.html

[73] Ladislav Bodnar (2003). *Linux distributions - facts and figures*. http://www.distrowatch.com/stats.php?section=packagemanagement

ANOTACIONES

[74] Barry W. Boehm (1981). *Software Engineering Economics*. Prentice Hall.

[75] *Legal resources and tools for surviving the bad patents*. http://www.bus-patents.com

[76] S. Bradner (oct.,1996). *The internet standards process – revision 3 (rfc 2026, bcp 9)*. http://www.ietf.org/rfc/rfc2026.txt

[77] Per Cederqvist and GNU (1993). *CVS–concurrent versions system*. http://www.gnu.org/manual/cvs/index.html

[78] Ward Cunningham. *Wiki design principles*

[79] Loic Dachary (2001). *Savannah the next generation*. http://savannah.gnu.org/docs/savannah-plan.html

[80] Junta de Andalucía. Decreto 72/2003 de 18 de marzo, de medidas de impulso de la sociedad del conocimiento en Andalucía, marzo, 2003. http://boja.andaluciajunta.es/boja/cgi-bin/frame_pagina.cgi?2003-55-8

[81] Adam de Boor. *Pmake – a tutorial*. http://docs.freebsd.org/44doc/psd/12.make/paper.html

[82] Miguel de Icaza. *The story of the gnome project*. http://primates.ximian.com/˜miguel/gnome-history.html

[83] Senado de la República Francesa. *Forum sur la proposition de loi tendant a généraliser dans l'administration l'usage d'internet et de logiciels libres*. http://www.senat.fr/consult/loglibre/index.htm

[84] Pedro de las Heras Quirós; Jesús M. González Barahona (2000). "Iniciativas de las administraciones públicas con relación al software libre". *Bole.TIC, revista de ASTIC* (vol. 14).

[85] Debian. *Debian free software guidelines*. http://www.debian.org/social_contract.html#guidelines

[86] Debian. *Debian policy manual*. http://www.debian.org/doc/debian-policy

[87] Debian. *Debian social contract*. http://www.debian.org/social_contract.html

[88] Schriftenreihe der KBSt (julio, 2003). *Leitfaden für die migration von basissoftwarekomponenten auf server- und arbeits-platzsystemen*. Technical

ANOTACIONES

report, Koordinierungs-und Beratungsstelle der Bundesregierung für Informations-technik in der Bundesverwaltung (KBSt). http://www.kbst.bund.de/download/mlf_v1_de.pdf

[89] Chris DiBona; Sam Ockman; Mark Stone (ed.) (1999). *Open Sources. Voices from the Open Source Revolu-tion*. O'Reilly & Associates. http://www.oreilly.com/catalog/opensources

[90] Open directory project. http://dmoz.org

[91] Justin R. Ehrenkrantz (mayo, 2003). "Release management within open source projects". En: *Proceedings of the 3rd Works-hop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. USA: Portland.

[92] Consejo Europeo (1991). Directiva 91/250/cee del consejo, de 14 de mayo de 1991, relativa a la protección jurídica de programas de ordenador. http://europa.eu.int/scadplus/leg/es/lvb/l26027.htm

[93] Karl Fogel; Moshe Bar (2001). *Open Source Code Development with CVS* (2.ª ed.). Paragliph Press. http://cvsbook.red-bean.com

[94] Electronic Frontier Foundation. *Open audio*. http://www.eff.org/IP/Open_licenses/eff_oal.html

[95] Free Software Foundation (1985). *The GNU manifesto*. http://www.gnu.org/philosophy

[96] Free Software Foundation (junio 1991). *Gnu general public license, version 2*. http://www.fsf.org/licenses/gpl.html

[97] Free Software Foundation (febrero, 1999). *Gnu lesser general public license, version 2,1*. http://www.fsf.org/licenses/lgpl.html

[98] Free Software Foundation. *Free software definition*. http://www.gnu.org/philosophy/free-sw.html

[99] Free Software Foundation. *Licencias libres*. http://www.gnu.org/licenses/license-list.html

[100] Bdale Garbee; Hartmut Koptein; Nils Lohner; Will Lowe; Bill Mitchell; Ian Murdock; Martin Schulze; Craig Small. *A brief history of debian*. En el paquete debian-history.

ANOTACIONES

[101] Daniel Germán (mayo, 2002). "The evolution of gnome". En: *Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering.* USA: Florida.

[102] Daniel Germán; Audris Mockus (mayo, 2003). "Automating the measurement of open source pojects". En: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering.* USA: Portland.

[103] Rishab A. Ghosh (marzo, 1998). "Cooking pot markets: an economic model for the trade in free goods and services on the internet". *First Monday* (vol. 3, núm. 3).
http://www.firstmonday.dk/issues/issue3_3/ghosh/index.html

[104] Rishab A. Ghosh; Ruediger Glott; Bernhard Krieger; Gregorio Robles (2002). *Free/libre and open source soft-ware: Survey and study - part iv: Survey of developers*.
http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf

[105] Rishab Aiyer Ghosh; Vipul Ved Prakash (julio, 2000). "The Orbiten free software survey". *First Monday* (vol. 7, núm. 5).
http://www.firstmonday.dk/issues/issue5_7/ghosh/index.html

[106] Michael W. Godfrey; Qiang Tu (agosto, 2000). "Evolution in open source software: A case study". *Proceedings of the 2000 International Conference on Software Maintainance.*

[107] Juan Alberto González (marzo, 2002). *Carta al congresista Villanueva*.
http://www.gnu.org.pe/mscarta.html

[108] Jesús M. González-Barahona; Miguel A. Ortuño Pérez; Pedro de las Heras Quirós; José Centeno González; Vicente Matellán Olivera (diciembre, 2001). "Counting potatoes: The size of Debian 2.2". *Upgrade Magazine* (vol. 6, núm. 2, págs. 60-66).
http://people.debian.org/˜jgb/debian-counting/counting-potatoes

[109] Michel Goosens; Sebastian Rahtz (1999). *The LaTeX Web Companion*. Addison Wesley.

[110] Burton Grad (enero-marzo, 2002). "A personal recollection: IBM's unbundling of software and services". *IEEE Annals of the History of Computing* (vol. 1, núm. 24, págs. 64-71).

[111] Working group on Libre Software (1999). *Free software / open source: Information society opportunities for europe?* http://eu.conecta.it/paper.pdf

ANOTACIONES

[112] GrULIC. *Legislación sobre el uso de software libre en el estado*. http://proposicion.org.ar/doc/referencias/index.html.es

[113] Jim Hamerly; Tom Paquin; Susan Walton (1999). *Freeing the source - the story of mozilla*. http://www.oreilly.com/catalog/opensources/book/netrev.html

[114] Michael J. Hammel (diciembre, 1991). "The history of xfree86". *Linux Magazine*. http://www.linux-mag.com/2001-12/xfree86_01.html

[115] S. Harris (agosto, 2001). *The tao of IETF: A novice's guide to the internet engineering task force (RFC 3160, FYI 17)*. http://www.ietf.org/rfc/rfc3160.txt

[116] Paul Harrison (2002). *The rational street performer protocol*. http://www.logarithmic.net/pfh/RSPP.

[117] Ragib Hasan. *History of Linux*. http://ragib.hypermart.net/linux

[118] Michael Hauben; Ronda Hauben (1997). *Netizens. On the History and Impact of Usenet and the Internet*. IEEE Computer Society Press.

[119] Kieran Healy; Alan Schussman (enero, 2003). *The ecology of open source software development*. http://opensource.mit.edu/papers/healyschussman.pdf

[120] Frank Hecker (mayo, 1998). *Setting up shop: The business of open-source software*. http://www.hecker.org/writings/setting-up-shop.html

[121] Frank Hecker (1998). *Setting up shop: The business of open-source software*. http://www.hecker.org/writings/setting-up-shop.html

[122] Guido Hertel; Sven Niedner; Stefanie Herrmann (2003). *Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel*. http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf

[123] Allan Heydon; Roy Levin; Timothy Mann; Yuan Yu (2001). *The vesta approach to software configuration management*. Technical report, COMPAQ Systems Research Center. ftp://gatekeeper.research.compaq.com/pub/DEC/SRC/research-reports/SRC-168.pdf

[124] Pekka Himanen (2001). *The Hacker Ethic and the Spirit of the Information Age*. Random House. http://www.hackerethic.org

ANOTACIONES

[125] Francis Hunt; Paul Johnson (2002). *On the pareto distribution of sourceforge projects*. Technical report, Centre for Technology Management, Cambridge University Engineering Department, Mill Lane, Cambridge CB2 1RX. http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf

[126] Open Source Initiative. *History of the osi*. http://www.opensource.org/docs/history.php

[127] Embajador de EE.UU. en Perú John R. Hamilton (junio, 2002). *Carta al presidente del congreso de la república*. http://www.gnu.org.pe/lobbyusa-congreso.html

[128] Paul Jones (mayo, 2000). *Brook's law and open source: The more the merrier?* http://www-106.ibm.com/developerworks/opensource/library/os-merrier.html?dwzone=opensource

[129] Niels Jorgensen. Incremental and decentralized integration in freebsd. En: Joe Feller; Brian Fitzgerald; Scott Hissam; Karim Lakhani (ed.) (2003). *Making Sense of the Bazaar*. O'Reilly.
http://www.dat.ruc.dk/˜nielsj/research/papers/bazaar-freebsd.pdf

[130] Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

[131] C. Kalt (abril, 2000). *Internet relay chat: Architecture (rfc 2810)*. http://www.ietf.org/rfc/rfc2810.txt

[132] J. Kelsey; B. Schneier (noviembre, 1998). "The street performer protocol". En: *Third USENIX Workshop on Electronic Commerce Proceedings*. USENIX Press. http://www.counterpane.com/street_performer.html

[133] J. Kelsey; B. Schneier (junio, 1999). "The street performer protocol and digital copyrights". *First Monday* (vol. 6, núm. 4).
http://www.firstmonday.dk/issues/issue4_6/kelsey

[134] Christopher M. Kelty (diciembre, 2001). "Free software/free science". *First Monday* (vol. 12, núm. 6).
http://firstmonday.org/issues/issue6_12/kelty/index.html

[135] Jamil Khatib. *Openipcore hardware general public license*.
http://www.opencores.org/OIPC/OHGPL_17.shtml

[136] Donald Knuth (1989). *The TeXboak*. Addison Welsley.

ANOTACIONES

[137] Stefan Koch (ed.) (2003). *Free/Open Source Software Development*. Idea Group Inc. http://wwwai.wu-wien.ac.at/˜koch/oss-book

[138] Stefan Koch; Georg Schneider (2000). "Results from software engineering research into open source development projects using public data. Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft". En: H.R. Hansen; W.H. Janko (Hrsg.). *Wirtschaftsuniversität Wien* (núm. 22).

[139] Sandeep Krishnamurthy (mayo, 2002). "Cave or community? an empirical examination of 100 mature open source projects". *First Monday* (vol. 4, núm. 7).
http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/index.html

[140] Laffitte, Trégouet; Cabanel (1999) *Proposition de loi numéro 495*. Senado de la República Francesa.
http://www.senat.fr/consult/loglibre/texteloi.html

[141] Laffitte; Trégouet; Cabanel (2000). *Proposition de loi numéro 117*. Senado de la República Francesa.
http://www.senat.fr/consult/loglibre/texteloi.html

[142] Leslie Lamport (1994). *LaTeX user's guide and reference manual*. Addison Welsley, Reading, Mass. (2.ª ed.).

[143] David Lancashire (diciembre, 2001). "Code, culture and cash: The fading altruism of open source development". *First Monday* (vol. 12, núm. 6).
http://www.firstmonday.dk/issues/issue6_12/lancashire/index.html

[144] M.M. Lehman; J.F. Ramil; P.D. Wernick (noviembre, 1997). "Metrics and laws of software evolution - the nineties view". *Proceedings of the 4th International Symposium on Software Metrics*.
http://www.ece.utexas.edu/˜perry/work/papers/feast1.pdf

[145] Barry M. Leiner; Vinton G. Cerf, Robert E. Kahn; David D. Clark; Leonard Kleinrock; Daniel C. Lynch; Jon Postel; Larry G. Roberts; Stephen Wolff (1997). *A brief history of the internet. Communications of the ACM*.
http://www.isoc.org/internet/history/brief.shtml

[146] Netcraft Ltd. (agosto, 2003) web server survey, 2003. http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html

[147] Mark Lucovsky (2000). *From nt os/2 to windows 2000 and beyond - a software-engineering odyssey*.
http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html

ANOTACIONES

[148] Gary McGraw. "Building secure software: How to avoid security problems the right way". En: David A. Wheeler http://www.dwheeler.com/sloc

[149] Marshall Kirk McKusick. "Twenty Years of Berkeley Unix. From AT&T-Owned to Freely Redistributable". En: DiBona y otros (1999). http://www.oreilly.com/catalog/opensources

[150] SUN Microsystems (2000). *Sun microsystems announces availability of StarOffice(TM) source code on OpenOffice. org*. http://www.collab.net/news/press/2000/openoffice_live.html

[151] Peter Miller (1993). *Aegis a project change supervisor*. http://aegis.sourceforge.net/auug93.pdf

[152] Audris Mockus; Roy T. Fielding; James D. Herbsleb (junio, 2000). "A case study of open source software development: the apache server". En: *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000, págs. 263–272)*. Irlanda: Limerick. ACM Press.

[153] Brian Molenaar. *What is the context of charityware?* http://www.moolenaar.net/Charityware.html

[154] *Mit opencourseware*. http://ocw.mit.edu

[155] Laurence W. Nagel (sept, 1996). *The life of SPICE*. Presented at the 1996 Bipolar Circuits and Technology Meeting. USA: Minneapolis, MN. http://www.icsl.ucla.edu/aagroup/Life%20of%20SPICE.html

[156] Alessandro Narduzzo; Alessandro Rossi (mayo, 2003). *Modularity in action: Gnu/Linux and free/open sour-ce software development model unleashed*. http://opensource.mit.edu/papers/narduzzorossi.pdf

[157] Nathan Newman (1999). *The origins and future of open source software*. http://www.netaction.org/opensrc/future

[158] Nupedia. http://www.nupedia.com

[159] Edgar Villanueva Núñez (2002). *Carta a Microsoft Perú*. http://www.gnu.org.pe/rescon.html

[160] Open Source Initiative. *Licencias de fuente abierto*. http://www.opensource.org/licenses/index.html

[161] Vilfredo Pareto. *Course of Political Economy*. Lausanne, 1896

ANOTACIONES

[162] Bruce Perens and The Open Source Initiative. (1998). *The open source definition*. http://www.opensource.org/docs/definition_plain.html

[163] GNU Perú. Proyectos ley software libre en la administración pública del gobierno peruano, congreso de la república.
http://www.gnu.org.pe/proleyap.html

[164] Walter Pinheiro. (diciembre, 1999). Proposicao pl-2269/1999: Dispoe sobre a utilizacao de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administracao pública. Câmara dos Deputados do Brasil. http://www.camara.gov.br/Internet/sileg/Prop_Detalhe.asp?id=17879, http://www.fenadados.org.br/software.htm

[165] Joseph Pranevich. (2003). *The wonderful world of Linux 2.6*.
http://www.kniggit.net/wwol26.html

[166] The Debian Project. *Debian developer map*.
http://www.debian.org/devel/developers.loc

[167] Joan Puigcercós Boixassa. (2002). Proposición de ley de medidas para la implantación del software libre en la administración del estado.
http://www.congreso.es/public_oficiales/L7/CONG/BOCG/B/B_244-01.PDF

[168] Joshua Quittner; Michelle Slatalla. (1998). *Speeding the Net: The Inside Story of Netscape and How It Challenged Microsoft*. Atlantic Monthly Pr.

[169] Chris Rasch. *A brief history of free/open source software movement*.
http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html

[170] Chris Rasch. (2001). "The Wall Street performer protocol. Using software completion bonds to fund open source software development". *First Monday* (vol. 6, núm. 6, mayo).

[171] Eric S. Raymond. *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates (http://www.ora.com), enero, 2001. http://catb.org/˜esr/writings/cathedral-bazaar

[172] Christian Robottom Reis; Renata Pontin de Mattos Fortes. (febrero, 2002). *An overview of the software engineering process and tools in the mozilla project*. http://opensource.mit.edu/papers/reismozilla.pdf

[173] Francois-René Rideau. (2000). *Patents are an economic absurdity*.
http://fare.tunes.org/articles/patents.html

ANOTACIONES

[174] Lawrence Roberts. (1978). "The evolution of packet switching". *Proceedings of the IEEE* (vol. 66, noviembre).

[175] Gregorio Robles; Jesús M. González Barahona; José Centeno González; Vicente Matellán Olivera; Luis Rodero Merino. (mayo, 2003). "Studying the evolution of libre software projects using publicly available data". En: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. USA: Portland.

[176] Gregorio Robles; Hendrik Scheider; Ingo Tretkowski; Niels Weber. (2001). *Who is doing it? knowing more about libre software developers*. http://widi.berlios.de/paper/study.pdf

[177] Marc Rochkind. (mayo, 1986). *Interview with dick haight*. *Unix Review*.

[178] Walt Scacchi. (2003). *Understanding open source software evolution: Applying, breaking and rethinking the laws of software evolution*. http://www.ics.uci.edu/˜wscacchi/Papers/New/Understanding-OSS-Evolution.pdf

[179] Bruce Schneier. (2000). *Software complexity and security*. http://www.counterpane.com/crypto-gram-0003.html

[180] Stephen J. Smoogen. *The truth behind red hat names*. http://www.smoogespace.com/documents/behind_the_names.html

[181] Haggen So. *Comparison of free/open source hosting (fosphost) sites available for hosting projects externally from project owners*. http://www.ibiblio.org/fosphost/exhost.htm

[182] Richard Stallman. *GNU coding standards*.
http://www.gnu.org/prep/standards.html

[183] Richard Stallman. *Why Free Software is better than Open Source*.
http://www.fsf.org/philosophy/free-software-for-freedom.html

[184] Richard Stallman. (1998). *Copyleft: Pragmatic idealism*.
http://www.gnu.org/philosophy/pragmatic.html

[185] Richard Stallman. (1998). *Why "free software" is better than "open source"*. http://www.gnu.org/philosophy/free-software-for-freedom.html

[186] Richard Stallman. (1998). *Why software should not have owners*.
http://www.gnu.org/philosophy/why-free.html

ANOTACIONES

[187] Richard Stallman. *The GNU Project*. En: DiBona y otros.
http://www.fsf.org/gnu/thegnuproject.html

[188] Richard Stallman. (junio, 1999). "On free hardware". *Linux Today*.
http://features.linuxtoday.com/news_story.php3?ltsn=1999-06-22-005-05-NW-LF

[189] Richard Stallman. (2001). *The free universal encyclopedia and learning resource*. http://www.gnu.org/encyclopedia/free-encyclopedia.html

[190] Richard Stallman. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Joshua Gay.

[191] Richard Stallman. (2003). *Some confusing or loaded words and phrases that are worth avoiding*.
http://www.gnu.org/philosophy/words-to-avoid.html

[192] Mitch Stoltz. (1999). *The case for government promotion of open source software*. http://www.netaction.org/opensrc/oss-report.html

[193] Andrew Tanenbaum; Linus Torvalds. (1999). *The tanenbaum-torvalds debate*. http://www.oreilly.com/catalog/opensources/book/appa.html

[194] Michael Tiemann. "Future of cygnus solutions. an entrepreneur's account". En: DiBona y otros.
http://www.oreilly.com/catalog/opensources/book/tiemans.html

[195] Linus Torvalds and David Diamond. (2001). *Just For Fun: the story of an accidental revolutionary*. Texere.

[196] Ilkka Tuomi. (2002). *Evolution of the Linux credits file: Methodological challenges and referen-ce data for open source research*. http://www.jrc.es/˜tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf

[197] Varios autores. *Carta abierta al director de la wipo*.
http://www.cptech.org/ip/wipo/kamil-idris-7july2003.pdf

[198] Pere Vigo i Sallent; Ernest Benach i Pascual; and Josep Huguet i Biosca. (mayo, 2002). *Proposició de llei de programari lliure en el marc de l'administració pública de Catalunya*. http://www.parlament-cat.es/pdf/06b296.pdf
http://www.hispalinux.es/modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=49

[199] Edgar Villanueva Núñez. (diciembre, 2001). *Proyecto de ley software libre, número 1609*. http://www.gnu.org.pe/proley1.html

ANOTACIONES

[200] Edgar Villanueva Núñez; Jacques Rodrich Ackerman. (abril, 2002). *Proyecto de ley de uso de software libre en la administración pública, número 2485*. http://www.gnu.org.pe/proley4.html

[201] W3C. (2000). Extensible markup language (xml) 1.0 (2.ª edic.).

[202] Norman Walsh; Leonard Muellner; Bob Stayton. (2002). *DocBook: The Definitive Guide*. O'Reilly. http://docbook.org/tdg/en/html/docbook.html

[203] Larry Welke; Luanne Johnson. (1998). *How the icp directory began*. http://www.softwarehistory.org/history/Welke1.html

[204] David A. Wheeler. (julio, 2000). *Estimating Linux's size*. http://www.dwheeler.com/sloc

[205] David A. Wheeler. (junio, 2001). *More than a gigabuck: Estimating gnu/Linux's*. http://www.dwheeler.com/sloc

[206] Eric Wiesstein. *Concise encyclopedia of mathematics*. http://mathworld.wolfram.com

[207] Wikipedia. *Gini coefficient*. http://www.wikipedia.org/wiki/Gini_coefficient

[208] Wikipedia. *Lorenz curve*. http://www.wikipedia.org/wiki/Lorenz_curve

[209] Wikipedia. *Pareto*. http://www.wikipedia.org/wiki/Pareto

[210] Wikipedia. *Tex*. http://www.wikipedia.org/wiki/TeX

[211] Brian Wilson. *Netscape navigator*. http://www.blooberry.com/indexdot/history/netscape.htm

[212] Computer World. (2000). *Salary survey 2000*. http://www.computerworld.com/cwi/careers/surveysandreports

[213] Robert Young. (1999). *Giving it away. how red hat software stumbled across a new economic model and helped improve an industry*. http://www.oreilly.com/catalog/opensources/book/young.html

[214] Jamie Zawinski. (1999). *Resignation and postmortem*. Why Mozilla project failed: http://www.jwz.org/gruntle/nomo.html

[215] Jamie W. Zawinsky. (1999). *Resignation and postmortem*. http://www.jwz.org/gruntle/nomo.html

ANOTACIONES

Appendix A. GNU Free Documentation License

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of *copyleft* which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

ANOTACIONES

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The *Document* below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as *you* You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A *Modified Version* of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A *Secondary Section* is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The *Invariant Sections* are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The *Cover Texts* are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A *Transparent* copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not *Transparent* is called *Opaque*.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The *Title Page* means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, *Title Page* means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section *Entitled XYZ* means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as *Acknowledgements*, *Dedications*, *Endorsements*, or *History*. To *Preserve the Title* of such a section when you modify the Document means that it remains a section *Entitled XYZ* according to this definition.

ANOTACIONES

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

ANOTACIONES

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the

Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.

- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.

- I. Preserve the section Entitled *History*. Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled *History* in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the *History* section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled *Acknowledgements* or *Dedications* Preserve the Title of the section, and preserve in the section all the

ANOTACIONES

substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

- M. Delete any section Entitled *Endorsements* Such a section may not be included in the Modified Version.

- N. Do not retitle any existing section to be Entitled *Endorsements* or to conflict in title with any Invariant Section.

- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled *Endorsements*, provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

ANOTACIONES

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled *History* in the various original documents, forming one section Entitled *History*; likewise combine any sections Entitled *Acknowledgements*, and any sections Entitled *Dedications*. You must delete all sections Entitled *Endorsements*.

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this

ANOTACIONES

License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an *aggregate* if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

ANOTACIONES

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled *Acknowledgements*, *Dedications*, or *History*, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License *or any later version* applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version

ANOTACIONES

number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the *with...Texts.* line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

ANOTACIONES

Apéndice B. Licencia de Documentación Libre de GNU

Ésta es una traducción no oficial de la GNU Free Document License (GFDL), versión 1.2 a Español (Castellano). No ha sido publicada por la Free Software Foundation y no establece legalmente los términos de distribución para trabajos que usen la GFDL (sólo el texto de la versión original en Inglés de la GFDL lo hace). Sin embargo, esperamos que esta traducción ayude los hispanohablantes a entender mejor la GFDL. La versión original de la GFDL esta disponible en la Free Software Foundation (http://www.gnu.org/copyleft/fdl.html).

Esta traducción está basada en una de la versión 1.1 de Igor Támara y Pablo Reyes. Sin embargo la responsabilidad de su interpretación es de Joaquín Seoane.

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. Se permite la copia y distribución de copias literales de este documento de licencia, pero no se permiten cambios.

B.1. PREÁMBULO

El propósito de esta Licencia es permitir que un manual, libro de texto, u otro documento escrito sea *libre* en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta Licencia proporciona al autor y al editor una manera de obtener reconocimiento por su trabajo, sin que se le considere responsable de las modificaciones realizadas por otros.

Esta Licencia es de tipo *copyleft*, lo que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Complementa la Licencia Pública General de GNU, que es una licencia tipo *copyleft* diseñada para el software libre.

ANOTACIONES

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: un programa libre debe venir con manuales que ofrezcan la mismas libertades que el software. Pero esta licencia no se limita a manuales de software; puede usarse para cualquier texto, sin tener en cuenta su temática o si se publica como libro impreso o no. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

B.2. APLICABILIDAD Y DEFINICIONES

Esta Licencia se aplica a cualquier manual u otro trabajo, en cualquier soporte, que contenga una nota del propietario de los derechos de autor que indique que puede ser distribuido bajo los términos de esta Licencia. Tal nota garantiza en cualquier lugar del mundo, sin pago de derechos y sin límite de tiempo, el uso de dicho trabajo según las condiciones aquí estipuladas. En adelante la palabra *Documento* se referirá a cualquiera de dichos manuales o trabajos. Cualquier persona es un licenciatario y será referido como *Usted*. Usted acepta la licencia si copia, modifica o distribuye el trabajo de cualquier modo que requiera permiso según la ley de propiedad intelectual.

Una *Versión Modificada* del Documento significa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma.

Una *Sección Secundaria* es un apéndice con título o una sección preliminar del Documento que trata exclusivamente de la relación entre los autores o editores y el tema general del Documento (o temas relacionados) pero que no contiene nada que entre directamente en dicho tema general (por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar nada de matemáticas). La relación puede ser una conexión histórica con el tema o temas relacionados, o una opinión legal, comercial, filosófica, ética o política acerca de ellos.

Las *Secciones Invariantes* son ciertas Secciones Secundarias cuyos títulos son designados como Secciones Invariantes en la nota que

ANOTACIONES

indica que el documento es liberado bajo esta Licencia. Si una sección no entra en la definición de Secundaria, no puede designarse como Invariante. El documento puede no tener Secciones Invariantes. Si el Documento no identifica las Secciones Invariantes, es que no las tiene.

Los *Textos de Cubierta* son ciertos pasajes cortos de texto que se listan como Textos de Cubierta Delantera o Textos de Cubierta Trasera en la nota que indica que el documento es liberado bajo esta Licencia. Un Texto de Cubierta Delantera puede tener como mucho 5 palabras, y uno de Cubierta Trasera puede tener hasta 25 palabras.

Una copia *Transparente* del Documento, significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público en general, apto para que los contenidos puedan ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por puntos) con programas genéricos de manipulación de imágenes o (para dibujos) con algún editor de dibujos ampliamente disponible, y que sea adecuado como entrada para formateadores de texto o para su traducción automática a formatos adecuados para formateadores de texto. Una copia hecha en un formato definido como Transparente, pero cuyo marcaje o ausencia de él haya sido diseñado para impedir o dificultar modificaciones posteriores por parte de los lectores no es Transparente. Un formato de imagen no es Transparente si se usa para una cantidad de texto sustancial. Una copia que no es *Transparente* se denomina *Opaca*.

Como ejemplos de formatos adecuados para copias Transparentes están ASCII puro sin marcaje, formato de entrada de Texinfo, formato de entrada de LaTeX, SGML o XML usando una DTD disponible públicamente, y HTML, PostScript o PDF simples, que sigan los estándares y diseñados para que los modifiquen personas. Ejemplos de formatos de imagen transparentes son PNG, XCF y JPG. Los formatos Opacos incluyen formatos propietarios que pueden ser leídos y editados únicamente en procesadores de palabras propietarios, SGML o XML para los cuáles las DTD y/o herramientas de procesamiento no estén ampliamente disponibles, y HTML, PostScript o PDF generados por algunos procesadores de palabras sólo como salida.

ANOTACIONES

La *Portada* significa, en un libro impreso, la página de título, más las páginas siguientes que sean necesarias para mantener legiblemente el material que esta Licencia requiere en la portada. Para trabajos en formatos que no tienen página de portada como tal, *Portada* significa el texto cercano a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del texto.

Una sección *Titulada XYZ* significa una parte del Documento cuyo título es precisamente XYZ o contiene XYZ entre paréntesis, a continuación de texto que traduce XYZ a otro idioma (aquí XYZ se refiere a nombres de sección específicos mencionados más abajo, como *Agradecimientos*, *Dedicatorias*, *Aprobaciones* o *Historia*. *Conservar el Título* de tal sección cuando se modifica el Documento significa que permanece una sección *Titulada XYZ* según esta definición.

El Documento puede incluir Limitaciones de Garantía cercanas a la nota donde se declara que al Documento se le aplica esta Licencia. Se considera que estas Limitaciones de Garantía están incluidas, por referencia, en la Licencia, pero sólo en cuanto a limitaciones de garantía: cualquier otra implicación que estas Limitaciones de Garantía puedan tener es nula y no tiene efecto en el significado de esta Licencia.

B.3. COPIA LITERAL

Usted puede copiar y distribuir el Documento en cualquier soporte, sea en forma comercial o no, siempre y cuando esta Licencia, las notas de copyright y la nota que indica que esta Licencia se aplica al Documento se reproduzcan en todas las copias y que usted no añada ninguna otra condición a las expuestas en esta Licencia. Usted no puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

Usted también puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

B.4. COPIADO EN CANTIDAD

Si publica copias impresas del Documento (o copias en soportes que tengan normalmente cubiertas impresas) que sobrepasen las 100, y la nota de licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible todos esos Textos de Cubierta: Textos de Cubierta Delantera en la cubierta delantera y Textos de Cubierta Trasera en la cubierta trasera. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como editor de tales copias. La cubierta debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede añadir otro material en las cubiertas. Las copias con cambios limitados a las cubiertas, siempre que conserven el título del Documento y satisfagan estas condiciones, pueden considerarse como copias literales.

Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la verdadera cubierta y situar el resto en páginas adyacentes.

Si Usted publica o distribuye copias Opacas del Documento cuya cantidad exceda las 100, debe incluir una copia Transparente, que pueda ser leída por una máquina, con cada copia Opaca, o bien mostrar, en cada copia Opaca, una dirección de red donde cualquier usuario de la misma tenga acceso por medio de protocolos públicos y estandarizados a una copia Transparente del Documento completa, sin material adicional. Si usted hace uso de la última opción, deberá tomar las medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio establecido por lo menos un año después de la última vez que distribuya una copia Opaca de esa edición al público (directamente o a través de sus agentes o distribuidores).

Se solicita, aunque no es requisito, que se ponga en contacto con los autores del Documento antes de redistribuir gran número de copias, para darles la oportunidad de que le proporcionen una versión actualizada del Documento.

ANOTACIONES

B.5. MODIFICACIONES

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las secciones 2 y 3 anteriores, siempre que usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto dando licencia de distribución y modificación de la Versión Modificada a quienquiera posea una copia de la misma. Además, debe hacer lo siguiente en la Versión Modificada:

- A. Usar en la Portada (y en las cubiertas, si hay alguna) un título distinto al del Documento y de sus versiones anteriores (que deberían, si hay alguna, estar listadas en la sección de Historia del Documento). Puede usar el mismo título de versiones anteriores al original siempre y cuando quien las publicó originalmente otorgue permiso.

- B. Listar en la Portada, como autores, una o más personas o entidades responsables de la autoría de las modificaciones de la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (todos sus autores principales, si hay menos de cinco), a menos que le eximan de tal requisito.

- C. Mostrar en la Portada como editor el nombre del editor de la Versión Modificada.

- D. Conservar todas las notas de copyright del Documento.

- E. Añadir una nota de copyright apropiada a sus modificaciones, adyacente a las otras notas de copyright.

- F. Incluir, inmediatamente después de las notas de copyright, una nota de licencia dando el permiso para usar la Versión Modificada bajo los términos de esta Licencia, como se muestra en la Adenda al final de este documento.

- G. Conservar en esa nota de licencia el listado completo de las Secciones Invariantes y de los Textos de Cubierta que sean requeridos en la nota de Licencia del Documento original.

ANOTACIONES

- H. Incluir una copia sin modificación de esta Licencia.

- I. Conservar la sección Titulada *Historia*, conservar su Título y añadirle un elemento que declare al menos el título, el año, los nuevos autores y el editor de la Versión Modificada, tal como figuran en la Portada. Si no hay una sección Titulada *Historia* en el Documento, crear una estableciendo el título, el año, los autores y el editor del Documento, tal como figuran en su Portada, añadiendo además un elemento describiendo la Versión Modificada, como se estableció en la oración anterior.

- J. Conservar la dirección en red, si la hay, dada en el Documento para el acceso público a una copia Transparente del mismo, así como las otras direcciones de red dadas en el Documento para versiones anteriores en las que estuviese basado. Pueden ubicarse en la sección *Historia*. Se puede omitir la ubicación en red de un trabajo que haya sido publicado por lo menos cuatro años antes que el Documento mismo, o si el editor original de dicha versión da permiso.

- K. En cualquier sección Titulada *Agradecimientos* o *Dedicatorias*, Conservar el Título de la sección y conservar en ella toda la sustancia y el tono de los agradecimientos y/o dedicatorias incluidas por cada contribuyente.

- L. Conservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Números de sección o el equivalente no son considerados parte de los títulos de la sección.

- M. Borrar cualquier sección titulada *Aprobaciones*. Tales secciones no pueden estar incluidas en las Versiones Modificadas.

- N. No cambiar el título de ninguna sección existente a *Aprobaciones* ni a uno que entre en conflicto con el de alguna Sección Invariante.

- O. Conservar todas las Limitaciones de Garantía.

Si la Versión Modificada incluye secciones o apéndices nuevos que califiquen como Secciones Secundarias y contienen material no co-

ANOTACIONES

piado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede añadir una sección titulada *Aprobaciones*, siempre que contenga únicamente aprobaciones de su Versión Modificada por otras fuentes –por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar.

Puede añadir un pasaje de hasta cinco palabras como Texto de Cubierta Delantera y un pasaje de hasta 25 palabras como Texto de Cubierta Trasera en la Versión Modificada. Una entidad sólo puede añadir (o hacer que se añada) un pasaje al Texto de Cubierta Delantera y uno al de Cubierta Trasera. Si el Documento ya incluye un texto de cubiertas añadidos previamente por usted o por la misma entidad que usted representa, usted no puede añadir otro; pero puede reemplazar el anterior, con permiso explícito del editor que agregó el texto anterior.

Con esta Licencia ni los autores ni los editores del Documento dan permiso para usar sus nombres para publicidad ni para asegurar o implicar aprobación de cualquier Versión Modificada.

B.6. COMBINACIÓN DE DOCUMENTOS

Usted puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia. Así mismo debe incluir la Limitación de Garantía.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y puede reemplazar varias Secciones Invariantes idénticas

ANOTACIONES

por una sola copia. Si hay varias Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título único de cada una de estas secciones añadiéndole al final del mismo, entre paréntesis, el nombre del autor o editor original de esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes de la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección Titulada *Historia* de los documentos originales, formando una sección Titulada *Historia*; de la misma forma combine cualquier sección Titulada *Agradecimientos*, y cualquier sección Titulada *Dedicatorias*. Debe borrar todas las secciones tituladas *Aprobaciones*.

B.7. COLECCIONES DE DOCUMENTOS

Puede hacer una colección que conste del Documento y de otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en todos los documentos por una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para cada copia literal de cada uno de los documentos en cualquiera de los demás aspectos.

Puede extraer un solo documento de una de tales colecciones y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraído, y siga esta Licencia en todos los demás aspectos relativos a la copia literal de dicho documento.

B.8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES

Una recopilación que conste del Documento o sus derivados y de otros documentos o trabajos separados e independientes, en cualquier soporte de almacenamiento o distribución, se denomina un *agregado* si el copyright resultante de la compilación no se usa para limitar los derechos de los usuarios de la misma más allá de lo que

ANOTACIONES

los de los trabajos individuales permiten. Cuando el Documento se incluye en un agregado, esta Licencia no se aplica a otros trabajos del agregado que no sean en sí mismos derivados del Documento.

Si el requisito de la sección 3 sobre el Texto de Cubierta es aplicable a estas copias del Documento y el Documento es menor que la mitad del agregado entero, los Textos de Cubierta del Documento pueden colocarse en cubiertas que enmarquen solamente el Documento dentro del agregado, o el equivalente electrónico de las cubiertas si el documento está en forma electrónica. En caso contrario deben aparecer en cubiertas impresas enmarcando todo el agregado.

B.9. TRADUCCIÓN

La Traducción es considerada como un tipo de modificación, por lo que usted puede distribuir traducciones del Documento bajo los términos de la sección 4. El reemplazo las Secciones Invariantes con traducciones requiere permiso especial de los dueños de derecho de autor, pero usted puede añadir traducciones de algunas o todas las Secciones Invariantes a las versiones originales de las mismas. Puede incluir una traducción de esta Licencia, de todas las notas de licencia del documento, así como de las Limitaciones de Garantía, siempre que incluya también la versión en Inglés de esta Licencia y las versiones originales de las notas de licencia y Limitaciones de Garantía. En caso de desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la nota de licencia o la limitación de garantía, la versión original en Inglés prevalecerá.

Si una sección del Documento está Titulada *Agradecimientos*, *Dedicatorias* o *Historia* el requisito (sección 4) de Conservar su Título (Sección 1) requerirá, típicamente, cambiar su título.

B.10.TERMINACIÓN

Usted no puede copiar, modificar, sublicenciar o distribuir el Documento salvo por lo permitido expresamente por esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o

ANOTACIONES

distribución del Documento es nulo, y dará por terminados automáticamente sus derechos bajo esa Licencia. Sin embargo, los terceros que hayan recibido copias, o derechos, de usted bajo esta Licencia no verán terminadas sus licencias, siempre que permanezcan en total conformidad con ella.

B.11.REVISIONES FUTURAS DE ESTA LICENCIA

De vez en cuando la Free Software Foundation puede publicar versiones nuevas y revisadas de la Licencia de Documentación Libre GNU. Tales versiones nuevas serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar nuevos problemas o intereses. Vea http://www.gnu.org/copyleft/.

Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que se aplica una versión numerada en particular de esta licencia o *cualquier versión posterior*, usted tiene la opción de seguir los términos y condiciones de la versión especificada o cualquiera posterior que haya sido publicada (no como borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada (no como borrador) por la Free Software Foundation.

B.12.ADENDA: Cómo usar esta Licencia en sus documentos

Para usar esta licencia en un documento que usted haya escrito, incluya una copia de la Licencia en el documento y ponga el siguiente copyright y nota de licencia justo después de la página de título:

Copyright (c) AÑO SU NOMBRE. Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos

ANOTACIONES

de Cubierta Trasera. Una copia de la licencia está incluida en la sección titulada *GNU Free Documentation License*.

Si tiene Secciones Invariantes, Textos de Cubierta Delantera y Textos de Cubierta Trasera, reemplace la frase *sin ... Trasera* por esto:

siendo las Secciones Invariantes LISTE SUS TÍTULOS, siendo los Textos de Cubierta Delantera LISTAR, y siendo sus Textos de Cubierta Trasera LISTAR.

Si tiene Secciones Invariantes sin Textos de Cubierta o cualquier otra combinación de los tres, mezcle ambas alternativas para adaptarse a la situación.

Si su documento contiene ejemplos de código de programa no triviales, recomendamos liberar estos ejemplos en paralelo bajo la licencia de software libre que usted elija, como la Licencia Pública General de GNU (*GNU General Public License*), para permitir su uso en software libre.

ANOTACIONES